ABSTRACT

Title: EXPLORING THE GEOGRAPHY OF ROUTINE ACTIVITY THEORY: A SPATIO-TEMPORAL TEST USING STREET ROBBERY

Elizabeth Ruth Groff, Ph.D., 2006

Co-Directed By: Ralph Dubayah, Professor, Geography
David Weisburd, Professor, Criminology and Criminal Justice

Many social phenomena have a spatio-temporal dimension and involve dynamic decisions made by individuals. Investigations focusing on the spatio-temporal dimensions of human behavior have received a great deal of theoretical attention; however, empirical testing of these theories has been handicapped by a lack of micro-level data and modeling tools that can capture the dynamic interactions of individuals and the context in which they occur. This research presents a methodology for evaluating theory through the implementation of a simulation model; the assumptions of the theory are operationalized in a model, a series of experiments are run, and the outcomes are analyzed to discover if they match what the theory would predict.

Specifically, the concepts of routine activity theory (RAT) (Cohen and Felson, 1979) are formalized in a computational laboratory representing Seattle, Washington. The computational environment for implementation, Agent Analyst, merges agent-based modeling (ABM) software with geographic information systems (GIS). A strategy for developing activity spaces is implemented and demonstrates how agents can move along existing street networks, and land use patterns can be used to create representational activity spaces.  Three versions of a model of street robbery are developed; each version implements a different level of constraints on agent's routine activities.  In one version (Simple), individuals are either at home or not at home.  In another, individuals follow a temporal schedule (Temporal).  Last, individual's schedules are both temporally and spatially constrained (Activity Space).  A series of experiments are conducted which compare the incidence and spatial pattern of street robbery events from each version.

The results of the experiments provide strong evidence of the important role routine activities play in street robbery events.  The addition of temporal and spatio-temporal schedule constraints reduces the incidence and changes the pattern of street robberies.  Support for routine activity theory's premise, as time spent away from home increases street robbery will increase, is found in the Simple and Temporal, but not the Activity Space version of the model.

# EXPLORING THE GEOGRAPHY OF ROUTINE ACTIVITY THEORY: A SPATIO-TEMPORAL TEST USING STREET ROBBERY

By

Elizabeth Ruth Groff

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

## Dedication

This research is dedicated to my grandmother Minnie Boltz who taught me to pursue greater understanding in all things and to Jo Fraley for her good humor and unwavering faith.

## Acknowledgements

There were a variety of University of Maryland faculty members and students who were instrumental in the completion of this research. Ralph Dubayah inherited my candidacy willingly and gave me lots of good advice. David Weisburd encouraged me and provided a skeptics view on the research. Jochen Albrecht guided my exploration of human spatial behavior, discussed details of the model implementation, and peer-reviewed the model code; his input was invaluable. In addition, the powerful computer Jochen loaned me was very useful. The other members of the committee offered their comments and participation for which I am grateful.

In general, the faculty at UMD in both the Geography and Criminology programs taught courses that were influential in expanding my horizons. In particular, Catherine Dibble's class on computational laboratories provided seminal guidance on how to specify an agent-based model. A special thanks to Femke Reitsma for listening, understanding, and challenging. Finally, Mel Songer took the journey with me, thanks for being there.

The author gratefully acknowledges partial financial support from the National Institute of Justice grant #2005-IJ-CX-0015. The Institute for Law and Justice's flexibility with regard to scheduling enabled my pursuit of this degree. Tom McEwen gave me helpful feedback on Chapter 3. Discussions with Kelly Bradley and Randall Guynes strengthened the research and my communication of it.

During the programming of the model, Jo Fraley's patient tutelage was invaluable to a novice programmer. Thanks to Tobi Glensk for building a prototype model in

Repast Java from my specifications.  Even though I ended up using different software, I learned a great deal about how to think like a programmer from my interaction with him.

This section would be incomplete without acknowledging my mentors from the University of North Carolina at Charlotte.  Sally Ives, who introduced me to behavioral geography and quantitative methods; Owen Furuseth who supported my desire to explore the geographical aspects of crime; and the rest of the faculty at UNC Charlotte who contributed in various ways -- Dennis Lord, Tink Moore, Wayne Walcott, Paul Smith, John Sommers, Gerry Ingalls and the Jerry Pyle.

Lastly, Julie Wartell provided a thorough edit for which I am very grateful.

In general:
To the many mentors/teachers/friends that have encouraged me over my lifetime, your words made a difference.

Thanks to my family for long distance support and to my friends for being patient and supporting my goals.

Finally, although she will never read this, my eternal gratitude goes to Katie Jane Groff for just being there.

# Table of Contents

# List of Tables

## List of Figures

# List of Maps

# Chapter 1: Introduction

"A temporal pattern is apparent in each and every spatial pattern (and) space and time are separable from one another only in abstraction." (Hawley, 1950 288)

Drawing on human ecological theory, Cohen and Felson put forward a "routine activity approach" to studying crime (Cohen & Felson, 1979 588). They hold that "criminal acts require the convergence in space and time of *likely offenders*, *suitable targets* and the *absence of capable guardians* against crime" (Cohen & Felson, 1979 588). According to the routine activity approach, convergence of individuals is the key dynamic element that enables illegal acts to occur during the course of everyday activities. Changes in social structure impact the frequency with which these elements converge by modifying the routine activity patterns of offenders, victims, and potential guardians. From this basic argument, they go on to hypothesize that if the frequency with which these elements converge in space and time increases, crime will also increase even if the supply of offenders or targets remains constant within a city. In this way, *routine activity theory* clearly states the basic elements necessary for a crime to occur and then elegantly ties the frequency with which those elements converge in space and time to macro level crime rates. Their intriguing argument recognizes the importance of space-time activities of individuals and makes the theory a logical choice on which to base this investigation.

Routine activity theory has been widely applied in empirical research and the theoretical framework is well-developed. However, because of difficulties in obtaining individual-level data and shortcomings in available statistical techniques, the empirical validity of the theory is still in question (Akers, 2000; J. E. Eck, 1995a). The inability to collect individual-level data, to characterize human travel behavior in general and the situational elements of crime events in particular, is an on-going barrier (Huisman & Forer, 1998; O'Sullivan & Haklay, 2000). Likewise, the identification of modeling tools capable of capturing the dynamic nature of human activities and interactions of individuals when they converge remains a hurdle.

In response to these challenges, some researchers have turned to simulation modeling as an alternative approach. Although modeling in general has a long been applied to examine social science phenomena, simulation has not (Gilbert & Terna, 1999; Gilbert & Troitzsch, 1999; R. G. Golledge, 1983; Harvey, 1969). Recent software developments have made access to simulation modeling much easier. Given the relative obscurity of simulation research, some background explaining how simulation models fit within a typology of modeling approaches is offered next.

In general, models offer a simplified representation of reality by attempting to capture only the most important elements of the phenomenon under study. Ostrom (1988) identified three types of models: verbal arguments, mathematical, and simulation. Verbal arguments describe relationships using words, where mathematical models use symbols and numbers. Mathematical models emphasize formalization and specificity over the richness of detail found in verbal models. However, complex mathematical models can quickly become computationally impractical requiring simplifying assumptions (e.g. rational man) to be used. The simplifying assumptions underpinning models are at the root of many debates in the literature and are critical to the veracity of the model created. When simplifying assumptions are incorrect, the efficacy of model results is compromised. Statistical models are a type of mathematical model representing the relationship among two or more characteristics of the unit of analysis (e.g. individuals, streets or cities). The implementation of statistical models in a wide variety of software packages has aided their adoption by researchers.

Finally, simulation models are similar to mathematical models in that they are developed from theory and involve a simplification of reality but their specification is in the form of a computer program (Gilbert & Terna, 1999; Ostrom, 1988). Once built, the computer program is run to obtain results, which are then analyzed using statistical models. Simulation models rely on a bottom-up approach. In other words, a few simple, theoretically based rules are developed for the individual agents. The interactions of individuals in the model produce the macro-level patterns that *emerge*. The property of *emergence* refers to any unexpected consequences from the application of simple behaviors (Epstein & Axtell, 1996; Gilbert & Troitzsch, 1999). In addition, individuals in simulation models are able to make dynamic decisions based on changing information (Bonabeau, 2002). Since routine activity theory posits both the micro-level factors necessary for a crime and the macro level consequences of changes in the convergence of those factors, simulation modeling is particularly well suited to operationalizing it.

The three types of symbol systems can be applied in the same research endeavor, and may even be at their most powerful when used in that way (Ostrom, 1988; Troitzsch, 1998). Accordingly, this research relies on all three symbol systems with different symbol systems playing a dominant role as the phases of the research progress. In the early stages, verbal models are used to describe what is known or thought to be known about human behavior in general and criminal decision-making within its environmental context in particular. These verbal descriptions are then formalized in the behavior rules of the model. In some cases, mathematical equations are embedded within the computer program to specify the relative importance of variables in the decision-making process. The environment and the people within that environment are defined within the program. The program also keeps track of the model time and the interactions among agents and their environment and writes out data at user-designated intervals. Statistical equations are then used to evaluate the results of the model runs. In this way, each of the symbol systems can be used to its best advantage with the goal of increasing the level of understanding of the process and/or relationships under study.

Agent-based models are a type of simulation model that consists of a collection of autonomous entities implemented within a software program. Entities in the model (i.e. agents) can represent people, governments, neighborhoods etc. Each entity has a set of unique characteristics and behaviors. Typically, these agents are placed in an artificial world to interact although there is a recent movement to use geographic information systems to provide a 'real' landscape (Brown, Riolo, Robinson, North, & Rand, 2005; O'Sullivan & Haklay, 2000). Agent-based models allow heterogeneity among individuals that more closely approximates the variety found in life. In addition, they are better able to accommodate the non-linearity in relationships that is frequently evident in complex and dynamic interactions (Dibble, 2003; Epstein & Axtell, 1996; Gilbert & Terna, 1999; Liu, Wang, Eck, & Liang, 2005).

Agent-based models can be implemented in the form of a computational laboratory. A computational laboratory is a set of software tools that enable the specification and execution of systematic experiments using simulation (Chen, Cunningham, Ewing, Peralta, & Visser, 1994; Dibble, 2001; Parker, Berger, & Manson, 2001; Slavin, 1996; Tesfatsion, 2001). An agent based modeling (ABM) simulation implemented in the framework of a computational laboratory offers the advantage of being able to hold the agents and/or the landscape constant and then vary one or both of them systematically. This feature provides a level of control difficult to attain using traditional social science methods (Dibble, 2003; Epstein & Axtell, 1996; Gilbert & Terna, 1999). The combination of heterogeneous agents and control enables the researcher to conduct a variety of experiments, using different conditions or applying various prevention scenarios, and then evaluate outcomes for minimal cost as compared to experiments undertaken in the real world. These characteristics directly address the shortcomings of earlier research testing routine activity and suggest ABM as an important component of a new, more flexible methodology.

The use of simulation models is not without its drawbacks. Like mathematical models, they are constrained by the original assumptions and the rules on which the model is based. This drawback is mitigated, but not eradicated, by the use of empirically-based parameters whenever possible. Relatedly, the creation of an artificial society opens the research effort to a variety of issues regarding the generalizability and usability of findings. Even when the simulation is based on a real place, the society is still representative rather than empirical. In addition, as compared to mathematical models, simulation does not produce any measures of the robustness of a particular solution (Axtell, 2000; Lempert, 2002). Mathematical models produce such statistics as confidence intervals that communicate the surety with which results can be interpreted. Simulation results can approach this type of measure through an iterative process of varying input parameters and cataloguing the outcomes of those variations so that we can begin to say something about the robustness of a particular model. Since the goal of agent-based modeling (ABM) is often directed toward explanation rather than prediction, the knowledge gained from using an iterative strategy to study artificial societies can serve to increase our understanding of how a process 'works' using the aforementioned strategy.

The employment of ABM in the social sciences has increased over the last ten years (Gilbert & Doran, 1994; Gilbert & Terna, 1999; Gilbert & Troitzsch, 1999; Macy & Willer, 2002). This rising interest in ABM stems from its unique capabilities which range from description, to knowledge discovery, to hypothesis testing. Another intrinsic advantage is derived from the computer code written for the simulation. Formalized computer code provides concrete documentation for the assumptions of the model and enables transparency in the research enterprise that is necessary for replication and testing of results (Chattoe 1996; Gilbert & Terna 1995). These attributes are especially important when trying to discover the mechanisms through which observed macro level patterns are formed. Criminologists have recently begun to explore how agent-based models can inform the study of crime (P. L. Brantingham & Brantingham, 2004; P. L. Brantingham & Groff, 2004; J. E. Eck & Liu, 2004; Xue & Brown, 2003). However, the inherently spatial nature of human movement and interaction, as well as the role of place in influencing those elements, require that models incorporate space as well as time.

Recent advances in technology have enabled the creation of the software packages necessary to enable agents to be 'situated' in a particular spatio-temporal milieu (e.g., agents can interact with data describing the streets in real environment). One example is the software product Agent Analyst which links two popular geographic information systems (GIS) and ABM software packages, ArcGIS (ESRI, 2005) and RepastPy (North, Collier, & Vos, 2006). The combination of the strengths of ABM and GIS is necessary in order to move away from the use of artificial landscapes and instead model individuals in their environment (Albrecht, 2005; An, Linderman, Qi, Shortridge, & Liu, 2005; Brown et al., 2005). The technological advances just discussed provide the tools for addressing the limitations of previous research. They facilitate the creation of a GIS/ABM model that is capable of capturing and analyzing: 1) the process involved in the convergence of offenders, victims, and guardians at a particular place and time; 2) the interaction that takes place once they occur; and 3) the culmination of those interactions in the form of emerging crime patterns.

## *1.0 Goal and Objectives*

This research proceeds in the tradition of Schelling (1971), Epstein and Axtell (1996) and many others where the goal of simulation is greater understanding rather than prediction. The primary goal of this study is to demonstrate how formalizing theory in a computational laboratory can provide a better understanding of how the spatio-temporal aspects of human activity influence the incidence and distribution of street robbery events. Accordingly, the point of this research is not to predict the pattern of street robbery events in Seattle, Washington, but rather to operationalize the assumptions of routine activity theory in an artificial society and then test whether the model outcomes match the predicted outcomes. The approach taken here emphasizes theory-testing but still in a theoretical world. In this way, the method represents an interim testing ground between the verbal formulation of the theory and the testing of theory with empirical data. While this exercise does not result in a determination of whether a theory is true in the real world, it does provide a way to test the plausibility of the theory's assumptions. To this end, the most parsimonious model possible is created, run, and the results subject to rigorous testing.

The choice to focus on a single crime is based the widespread recognition that crime is not a homogenous phenomenon. Narrowing the study to only one type of crime aids in the interpretation of findings and subsequently, to better understanding (Clarke, 1983; Clarke & Cornish, 1985;2001; Zahn & Jamieson, 1997). The crime of street robbery offers several advantages for this study. First, it is an instrumental crime and thus more likely than expressive crimes to involve a rational decision process (Clarke & Cornish, 1985; Cornish & Clarke, 1986; Walsh, 1986). Second, street robbery is by definition restricted to the street or some other exposed area rather than in a residence or business and thus involves the public intersection of offender and target in space and time. Third, police presence is assumed to be more effective against street level crime then crimes that take place indoors (e.g. domestic violence). Fourth, street robbery elicits a high level of fear among residents because of its suddenness and potential for serious injury and thus is of considerable interest to both law enforcement and the public (Feeney, 1986).

The following objectives are met in order to achieve the overall goal.

1) Formalize routine activity theory (Cohen & Felson, 1979) in a GIS/ABM so that the dynamics of individual level decision-making and behavior that produce macro level street robbery patterns can be represented.
2) Explore the role of geography (i.e. activity spaces) by having the agents interact in a 'real' environment, and by making the spatio-temporal dimension of human behavior explicit in the model and the experiments.
3) Create and run a series of experiments to test whether: a) the theoretically-predicted outcomes from routine activity theory match the model-produced outcomes; and b) changing the spatio-temporal schedules of individuals produces different quantities and spatial distributions of street robbery.

As the initial foray into this area, the first objective of the study focuses on the development of several versions of a GIS-enabled computational laboratory for modeling some simple, dynamic interactions between individuals from which aggregate crime rates and patterns of crime emerge. The modeling approach involves a multi-step process that begins with a review of the published writings on routine activity theory to identify and formalize key concepts in the theory. Those key concepts are then incorporated into the models in the form of behavior rules. Because opportunity theories focus on dynamic, individual-level interactions within a particular environmental context, the application of a geographic perspective is a positive, and I would argue necessary, dimension to modeling crime. Specifically, a geographic perspective makes explicit the role of the distribution of land uses and the shape of transportation systems in human movement. Human movement is critical to understanding convergence (i.e. *why* suitable targets and motivated offenders converge *where* and *when* there is a lack of capable guardians).

The creation and testing of the model is completed over three phases: 1) development of a conceptual model of crime; 2) implementation of the base model; and 3) verification of the base model. The initial phase relies on a careful examination of the existing theory and empirical results to inform the identification and definition of relationships used in the model. The result of this step is a conceptual model specifying the important constructs that underlie routine activity theory. These constructs are operationalized through the identification of specific variables and how they will be measured (Kerlinger & Lee, 2000). Next, the base version of the model is programmed using the specifications from the conceptual model and making changes as needed because of the specific configuration of software tools chosen for implementation. The resulting computer program is systematically tested to ensure proper operation and the code verified. Extensive testing of the entire technological framework is conducted to ensure that the model performs as expected. Finally, experiments are conducted to answer the research questions. The same testing protocol is applied to each of the model versions as they are implemented under the third objective.

The second objective is to explore the role of geography by making the spatio-temporal dimension of human behavior explicit in the model. This objective is met by varying the spatio-temporal aspects of agent activity spaces across versions of the model. The Simple version of the model does not contain elements of spatial or temporal activity spaces; rather, the individuals have a defined amount of time to spend at home each day but the rest of the time they move randomly along a street network. This model acts as a null to which the other models are compared. Two additional versions incorporate the idea of temporally and spatially constrained activity spaces. In one, the individuals have a temporal activity program with random movement. In the other, the individuals have the same temporal activity program but travel among a set of activity locations. Together, the activity locations and the path taken among them constitute a defined routine activity space.

The third objective is met by using the computational laboratory developed to explore the following research questions: 1) Does the shift of routine activities away from home increase street robbery? 2) Does the spatial distribution of street robberies change as routine activities shift away from home? and 3) How does the spatio-temporal structure of routine activities influence the incidence and spatial pattern of street robbery? The same set of experiments is run for each of the model versions.

## *1.1 Expected Significance of the Research*

The proposed methodology offers several advances. First, and most generally, this work demonstrates the value of simulation for theory testing and exploration. This method is particularly pertinent in social science where the ideal of falsification of theories with empirical data is difficult to achieve (Popper, 1965). Falsification occurs when the systematic empirical observations do not match what the theory predicted (Vold, Bernard, & Snipes, 2002). But in order to be falsified, a theory must be “testable by objective, repeatable evidence” (Akers, 2000 7). One major challenge to establishing the empirical validity of routine activity theory is the lack of empirical data on the micro-level interactions of individuals in space-time. The methodology posited here provides an alternative to empirical studies that is both systematic and repeatable. The method also enables the use of rigorous research design principles in conducting experiments. The virtual laboratory is particularly attractive for theory testing when ethical or monetary reasons preclude the conduct of empirical research experiments. In addition, the methodology promotes rigor and specificity by requiring the formalization of theoretical concepts before translating those concepts into a simulation model and by codifying the assumptions of the model in the computer program which implements them. These characteristics provide a level of transparency that is necessary for repeatability and for comparison of different implementation models to one another.

Second, the research integrates the strengths of GIS and ABM in a computational laboratory enabling the combination of empirical data representing travel opportunities and situational context from a GIS with ABM's dynamic agent decision-making. As a result, direct representation and manipulation of individuals' routine activities is achievable. The combination also makes possible the simulation of the micro- and macro-level interactions in a crime event as it happens. The resulting spatio-temporal patterns can be displayed and analyzed. Third, the completed simulation model offers the first test of routine activity theory in which the individual-level decisions and the spatio-temporal aspects of activity spaces are quantified and the impacts of movement on the resulting crime patterns are measured.

Looking forward, these base versions of a street robbery model can serve as the foundation for subsequent extensions that will more richly represent both human behavior and context as they are related to crime. The results of experiments conducted with these and future versions of the model have the potential to inform the creation of crime prevention strategies and contribute to the body of knowledge in geography, criminology, and social sciences in general.

## *1.2 Organization of the Remaining Text*

The dissertation is organized in the following manner. A series of three papers form the body of the dissertation. I'll apologize up front for information that is repeated among the papers. Each paper/chapter represents one of the objectives just outlined. Chapter 2 details a new methodology for 'situating simulation' by developing activity spaces based on the land use of an existing city and implementing movement along a vector street network. Chapter 3 discusses the formalization of routine activity theory in a computational laboratory and reports whether the outcomes from a simple model reflect what the theory predicts. Chapter 4 extends the basic model of street robbery outlined in the second paper by incorporating the routine activity spaces and agent movement discussed in Chapter 2 in two new versions of the model and then compares the results from all three versions. Chapter 5 summarizes the findings, discusses potential limitations, and then puts forward directions for future research. Complete documentation of the model versions is provided in the Appendices. Appendix A contains the documentation for the simple random version and Appendix B the documentation for the two versions in which each individual has a defined schedule (one temporal and the other spatio-temporal).

Appendices C, D, and E consist of the software code for the three versions of the model. The print statements used for debugging are left in the code to aid subsequent users.

# Chapter 2: 'Situating' Simulation

## *1.0 Introduction*

The importance of incorporating space and time into research on human behavior has long been recognized in a variety of disciplines (Chorley & Haggett, 1967; Engel-Frisch, 1943; Hägerstrand, 1973; Harvey, 1969; Hawley, 1950; Horton & Reynolds, 1971; H. J. Miller, 1991; R. J. Sampson, 1993). Addressing spatio-temporal interactions among individuals and their environments is a challenging undertaking with two major hurdles. First, it is not enough to be able to capture the multitude of individual decisions that occur within unique contexts; the method must also be able to accommodate dynamic changes in the characteristics of individuals and situations that influence the outcome of subsequent interactions. Second, individual-level data that can support these types of studies must be obtained. Although there has been tremendous growth in the availability of micro-level data describing places, data regarding the daily activities of individuals (e.g. the type, location, and duration of activity) remains sparse. While these data are essential to modeling the spatio-temporal convergence and interaction of individuals, they are unlikely to become available due to privacy concerns (O'Sullivan, 2004b).

Given the large quantity of micro level environmental data and mature software packages, researchers frequently turn to geographic information systems (GIS) to model human behavior (An et al., 2005; Kwan, 1998; H. J. Miller, 1991). GIS provide a powerful tool for collecting, managing and displaying the multitude of spatially explicit data available on places but they are unable to model the dynamic, individual-level interactions across time. The inability of GIS to accommodate time is a well-known issue that remains unsolved despite a great deal of attention (Albrecht, Forthcoming; Brown et al., 2005; Peuquet, 1994;2002). Physical scientists address this issue in their process models by preparing their data in a GIS and then analyzing them in a dynamic model (Maguire, Batty, & Goodchild, 2005).

Simulation modeling offers an alternative method capable of capturing dynamic interactions among individuals taking place at the micro level and their relationship to macro level patterns. All models, simulation or otherwise, involve the creation of a simplified representation of a social phenomenon (Gilbert & Terna, 1999). In the case of statistical models, such as a regression model, input data are 'run' via a statistical program which produces output data describing the relationships among the input data. Simulation models, in contrast, are computer programs themselves; programs that describe critical aspects of the social phenomenon being modeled. The program is run and the output data are analyzed using standard statistical techniques. Simulation modeling has three main advantages over statistical models. First, it allows heterogeneity among individuals that more closely approximates the variety found in life. Second, it is able to accommodate the non-linear relationships present in dynamic and complex interactions (Dibble, 2003; Epstein & Axtell, 1996; Gilbert & Terna, 1999). Third, simulation modeling can be used in situations where little or no data empirical data are available. Statistical models require data, either empirical or simulated.

Agent-based modeling (ABM) is one type of simulation that employs a bottom-up approach in which agents are imbued with unique characteristics and general behavioral rules (Epstein & Axtell, 1996; Gilbert & Terna, 1999; Gilbert & Troitzsch, 1999). An agent is an "autonomous goal-directed software entity" (O'Sullivan & Haklay, 2000) that most often represents a person but can also represent organizations, neighborhoods etc. The outcome of interaction with other agents is driven by the decisions of the individuals involved and those decisions dynamically change the characteristics of agents. These agents are removed from their 'real-word' situation and placed in an artificial world (O'Sullivan & Haklay, 2000). The use of artificial landscapes which do not take into account the impact of the environment in which individuals move and interact represents one significant drawback to ABM.

A natural application for ABM is toward achieving a better understanding of the crime event in its situational context. Some researchers have already begun to explore the use of simulation for capturing the dynamic interactions taking place at the micro level and their relationship to macro level patterns (P. L. Brantingham & Brantingham, 2003;2004; P. L. Brantingham & Groff, 2004; J. E. Eck & Liu, 2004; Gunderson & Brown, 2003; Wang, Liu, & Eck, 2004). However, these efforts rely on the use of artificial landscapes to inform agent activities and movement.

In order to model individuals in a non-artificial environment, an approach is needed that combines the strengths of ABM and GIS (Albrecht, 2005; An et al., 2005; Brown et al., 2005). A combined ABM/GIS simulation model integrates the advantages of autonomous agents found in agent-based modeling with the spatial explicitness of a geographic information system. This allows agents to interact on city streets and their activities during the simulation to be informed by the distribution of opportunities for housing, employment, shopping, and recreation across the urban backcloth. The result is agent spatial behavior that is more representative of actual human behavior than when agents are created with and interact on artificial landscapes.

This research documents the successful implementation of a model of street robbery in which agents move on the vector street network and their activity spaces reflect the distribution of population, jobs, and retail/services/recreation opportunities. The methodology is implemented using Agent Analyst software, an integrated ABM/GIS tool. The GIS component enables the creation of realistic activity spaces and the movement of agents along vector street networks, while the ABM controls the temporal elements of the simulation and the interaction of agents with one another. By combining the two, the situational elements of the convergence of offender and victim at a specific place and time are simulated. The crime of street robbery is a natural choice for this type of model because it stems from the interaction of individuals in a public area. In addition, it is an instrumental crime (for economic gain), and thus more likely to be the result of a rational decision than an expressive crime (Clarke & Cornish, 1985; Walsh, 1986). The remainder of this paper is organized as follows. First, the theoretical basis for the model is covered and a conceptual model presented. Next the implementation details including model rules and input data are provided. Lastly, the results and the implications for future research are discussed.

## *2.0 Theoretical Basis for a Street Robbery Model*

Both criminological and geographical knowledge are important to the development of a conceptual model of street robbery events. Opportunity theories of crime, ones that address the elements of the situation in which the offender makes the decision to offend, form the basis of the model.[1] Specifically, routine activity theory (RAT) (Cohen & Felson, 1979) provides the structure of the model and rational choice theory (Clarke & Cornish, 1985;2001) is used to guide offender decision making. The structure and timing of agent's activities and movement along the street network is informed by the major theoretical perspectives that address routine activity spaces.

## 2.1 Criminological Theory

Cohen and Felson's (1979) routine activity theory identifies the key to increases in crime as the shift of routine activities away from home. The theorists hypothesize that as individuals spend more time away from home, crime will increase. As originally conceptualized, routine activity theory identifies the convergence of *motivated offender*, *suitable target*, and the *lack of* a *capable guardian* at a particular place and time as the core elements necessary for a crime to occur. They emphasize that crimes occur when the normal everyday activities of offenders and victims intersect with no guardian present.

The theorists also recognize the importance of *routine activities* in influences when and where victims and offenders converge but they do not directly address the details of human mobility. They view routine activities as the key dynamic element in determining aggregate crime rates because it affects the three other elements

[1] Space constraints prohibit a detailed examination of extensions to routine activity theory or even related opportunity theories pertinent to micro level modeling. Rational choice theory is addressed because it provides for bounded rationality in the decision to offend. For a complete overview of theories and how they inform the criminal event perspective (CEP) please see (Meier, Kennedy, & Sacco, 2001). Several books offer a good overview of opportunity theories (Akers, 2000; Cullen & Agnew, 1999; Vold et al., 2002).

necessary for a crime, motivated offender, suitable target, and guardianship. Changes in routine activities directly impact the frequency of convergence among these elements which in turn, increase or decreases overall crime rates. Thus, the theorists neatly tie the interaction of clearly defined elements of a crime to societal level crime rates. These four elements of offender, target, guardian, and routine activities form the main constructs of the model.

As previously mentioned, routine activity theory pays little attention to the source of the offender's motivation and assumes a supply of motivated offenders. Consequently, the model developed here relies on rational choice theory for the specifics of offender decision-making (Clarke & Cornish, 1985). Rational choice theory is based on the economic principle of expected utility where each individual's decisions are predicated upon balancing projected benefits against projected costs of activities. The theory does not assume people have perfect knowledge but rather recognizes that offenders make the decision to commit a particular offense based on the characteristics of the specific situation using bounded rationality (i.e. imperfect knowledge) and taking into account three factors: the suitability of the situation, the presence of a viable target and the level of guardianship. Rational choice theory also assumes that offender spatial behavior is essentially similar to that of non-offenders so all people in the model who are not police have identical movement rules.

## 2.2 Activity Spaces Theory

One of the core concepts in routine activity theory involves the necessity of the convergence of victims and offenders in space and time. The specific 'where' and 'when' of convergence stems from the routine behavior patterns of each actor involved. Thus representing the spatio-temporal aspects of human behavior that facilitate convergence is a critical element in modeling street robbery events since it is the interactions between humans and their environment that serve as the source of explanation of observed spatial patterns (Aitken, Cutter, Foote, & Sell, 1989; Gold, 1980; R. G. Golledge & Timmermans, 1990; Timmermans & Golledge, 1990; Walmsley & Lewis, 1993).

A large quantity of research is available to inform agent movement and routine activities in the model and that research suggests people tend to have an area within which they conduct their daily activities. Some researchers term this area an *activity space* (Horton & Reynolds, 1971), some call it a *potential path area* (H. J. Miller, 1991), and others a *domain* (Hägerstrand, 1970;1975). This area encompasses both the locations that are visited and the paths taken among those locations. Different perspectives have their own terms for these locations and paths. Locations that are visited are called *stations* (Hägerstrand, 1970;1975), *nodes* (Paul Brantingham & Brantingham, 1981b; Patricia Brantingham & Brantingham, 1993; Lynch, 1960; H. J. Miller, 1991), or *anchorpoints* (R. Golledge & Stimson, 1997; R. G. Golledge, 1978). These are the places where the majority of human interaction occurs. The particular routes taken among the locations are termed *paths* (Hägerstrand, 1970;1975; Lynch, 1960). None of these elements are static, for example, the shape and size of areas (i.e. activity spaces) can change as people change jobs (i.e. nodes) or as their circumstances change (Hägerstrand, 1970).

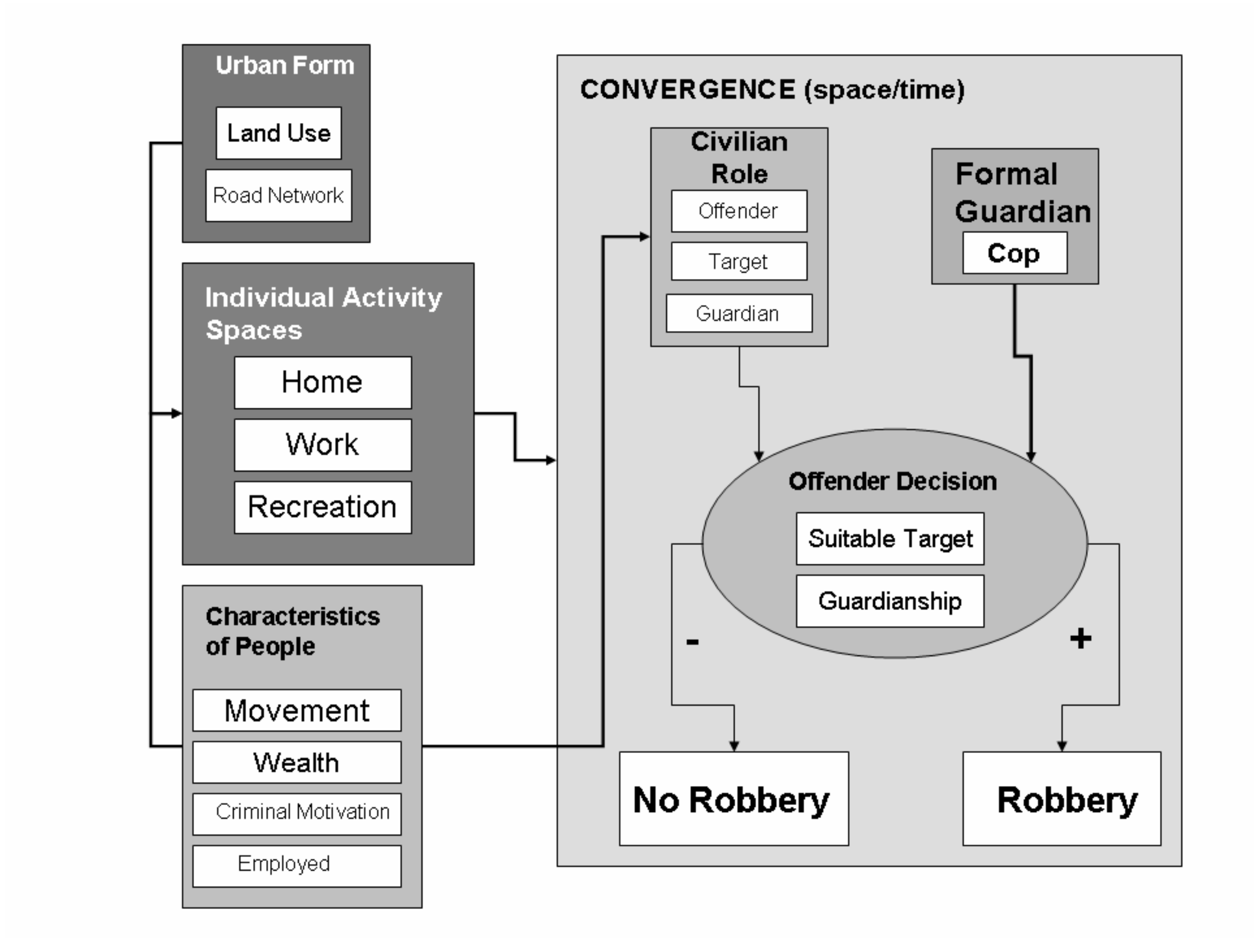
Regardless of the terminology, home tends to be the dominant place in any activity space. Travel tends to be concentrated along certain routinely frequented paths. Frequently traveled paths may be important factors in determining aggregate crime patterns because they bring offenders and victims together in space and time. Individual's travel patterns are influenced by constraints (i.e., temporal, economic and spatial) on their ability to take advantage of opportunities for housing, employment, recreation etc

Together this collection of research provides a strong basis for conceptualizing routine activity spaces of individuals as a set of places and the paths between those places. Specifically, agents in the model have four places they visit each day: a home, a main node (e.g. work, school, etc.), and at least two other places that are visited frequently such as a gym, grocery store, dry cleaner, class etc. The paths taken to travel among the places are structured by the street network.

### 2.3 Conceptual Model

The preceding review of research identifies the basic elements represented in the conceptual model (see boxes in Figure 2-1). The conceptual model identifies two classes of people, civilians and police. Civilians have activity spaces and can take on different roles (i.e., offender, victim, or guardian) depending on the particular situation. Police exist only as agents of formal guardianship. Civilians with criminal propensity can potentially take on any one of three roles, offender, victim or guardian. Civilians without criminal propensity can be either victims or guardians. In addition to criminal propensity, each civilian in the model has a unique set of characteristics that include wealth and employment status.

Figure 2-1: Conceptual Model of Street Robbery



Two other spatial elements are important to convergence of civilian agents in a model of street robbery. One is the activity spaces of the agents and the other is the network of streets available for travel. The size and form of activity spaces is influenced by the distribution of residential housing, jobs, schools, retail and services. Each civilian has a unique activity space reflecting the places they visit. Once convergence occurs, factors such as guardianship, and suitability of target are considered by the offender when making the decision whether or not to commit a robbery.

## *3.0 Research Design*

This section describes a methodology for 'situating' simulation models including software, data, movement and activity space formulation. Recent developments in technology and increased data availability at the micro-level support this approach to modeling individual-level phenomena. The move to object-oriented architecture provides the technical foundation for the integration of GIS and ABM. Section 3.1 provides specifics about the software package used to implement the methodology. Next, the data used to inform agent movement and the activity spaces of the agents is described. Section 3.3 explains how random and directed movement of agents is implemented in the model. Lastly, section 3.4 gives the details of how agent activity spaces are constructed so they reflect the actual distributions of homes, jobs and opportunities for retail, recreation, and services.

### 3.1 Agent Analyst- GIS/ABM Integration

The method uses a new software package, Agent Analyst, which integrates GIS and ABM to provide a platform for the dynamic modeling of individuals across space and time.[2] This package follows the middleware approach in which the temporal relationships are handled by the ABM software and the topological relationships are managed by the GIS (Brown et al., 2005). Agent Analyst combines two of the most popular packages for ABM and GIS, the Recursive Porous Agent Simulation Toolkit (Repast) and ArcGIS. To make the software easier to use, Agent Analyst is built using the rapid development version of Repast called Repast for Python Scripting

[2] Agent Analyst under development as a partnership between ESRI and Argonne National Laboratories; they are the parent companies of ArcGIS and Repast respectively. Agent Analyst is free but currently available by request only. The website for Repast is http://repast.sourceforge.net/.

(RepastPy) which has a graphical user interface that automates much of the programming to create the framework of a model. Agent Analyst is designed to be added into ArcGIS as a toolbox. Once the toolbox is added in ArcGIS, individual models can access shapefiles allowing: 1) individual agents to become spatially aware and 2) the visualization of agent movement and decision outcomes (e.g. locations of crimes).

The integration of GIS and ABM enables the exploration of how individual decisions by heterogeneous agents translate into aggregate rates of street robbery. ABM permits the researcher to: 1) collect data about the characteristics of each individual present during an interaction; 2) randomly assign characteristics to agents greatly reducing the possibility of systematic bias; 3) allow agents to make independent decisions within behavioral guidelines; and 4) systematically vary one attribute while holding all others constant to undertake controlled, repeatable experiments (Dibble, 2003; Epstein & Axtell, 1996; Gilbert & Terna, 1999). GIS makes it possible to take into account how the characteristics of the real environment (i.e. street network, distribution of homes, jobs and activities) impact the activity spaces of agents. In addition, it provides the ability explore the role of routine activities in facilitating the space-time convergence of a motivated offender and a suitable target, without a capable guardian present.

## 3.1 Data

The initial implementation of the model is situated in Seattle, Washington which provides the data for the model landscape and the agent activity spaces. Four input datasets describing conditions in Seattle are used to inform the activity spaces of agents in the model: 1) total population; 2) total employment; 3) total potential activities; and 4) streets. Blockgroup level population figures are used to describe the distribution of residences across Seattle (U.S. Census Bureau, 2000). Employment data are used to describe the number of employees per zip code area (U.S. Census Bureau, 2002). The 18,024 potential activity locations are identified through the use of retail and service establishments (e.g. grocery stores, convenience stores, dry cleaners, gyms etc.) (ESRI, 2003). The street network, derived from the King County Street Network Database (SND) file, is used to structure agent travel.

In addition to the input data describing Seattle, twelve parameters are set prior to the model run. The rationale for their initial settings is described in detail in Table 2-1. Random number seeds are a unique type of parameter used in the model; they provide the ability to replicate the model behavior over subsequent runs and are essential to using simulation as a laboratory for experimentation (Axelrod, Forthcoming).

Table 2-1: Parameters in the Model

| Variable | Rationale |
|---|---|
| **Society Level** | |
| Number of Agents = 1000 | Represents a balance between ensuring there are enough agents so that interactions can occur and the computational overhead from using more agents |
| Number of Police = 200 | Chosen to ensure that police agents would be present at some of the convergences that occur across the 16,035 places in Seattle. |
| Unemployment Rate = 6% | The unemployment rate of six percent is based on the 2002 unemployment rate for Seattle (Bureau of Labor Statistics, 2003).[3] |
| Rate of Criminal Propensity = 20% | Given that 20% of the population has committed a crime, 20% of civilians are assigned criminal propensity using a uniform distribution (Visher & Roth, 1986). |
| Time To ReOffend = 60 | Parameter value chosen as a starting point since the author could find no empirical data on which to base time to reoffend.. |
| Random Number Seed = 100 (seed also tested at 200, 300, 400 and 500) | An explicit random number seed based on the Mersenne Twister (MT) algorithm is used as the basis for all random number distributions used in the model. MT is currently considered to be the most robust in the industry (Ropella, Railsback, & Jackson, 2002). |
| **Agent Level** | |
| Societal Time Spent Away From Home = 30% (40%, 50%, 60%, 70%) | Assigned based on a normal distribution with a mean of 432 minutes (for the 30% condition) and a standard deviation of 10% of the mean (sd = 43).[4] |
| Initial Wealth = 50 | Initial wealth is assigned with a mean of 50 and a standard deviation of 20 units. |
| Amount of wealth received each payday = 5 | No empirical evidence available. |
| Amount of wealth exchanged during robbery=1 | No empirical evidence available.[5] |
| **Situation Level** | |
| Guardianship Perception = U(-2,2) | The guardianship perception value can add or subtract zero, one or two guardians from the actual number present. This represents the stochastic element in the offender's perception of the willingness of a guardian to intervene. |
| Suitable Target Perception = U(-1,1) | The value in suitable target can increase or decrease the suitability or leave it unchanged. This enables the offender to sometimes decide a target is not suitable even when they have more wealth. |

[3] Since the jobs data are from 2002, the corresponding year's unemployment rate is used.
[4] In Groff (Forthcoming-a) the time spent away from home is systematically varied to test the core proposition of routine activity that as time spent away from home increases crime will increase.
[5] A request to the Seattle Police Department for the average amount of cash taken during street robberies remains unanswered.

The outcome data from the simulation are collected for individual civilians, street nodes/places and for the society as a whole (Table 2-2). Data are collected at intervals during the model and at the completion of each model run. These data are written to two types of files, text files and shapefiles. In a simulation model, the modeler controls the data that are collected and how frequently they are written to a file. There is a computational cost each time the program writes to a file that must be balanced with the need for information about the model run.

Table 2-2: Outcome Data from Model

| **Variable Name** | **Description** | **Level of Measurement** |
|---|---|---|
| **Societal-level Outcome** | | |
| TotRob | Total number of robberies | Ratio |
| RobRate | Average number of robberies per population | Ratio |
| TotConverge | Total number of convergences (i.e. situations with a motivated offender and one or more 'at risk' civilians) | Ratio |
| TotDeterred | Total number of robberies deterred by a cop's presence | Ratio |
| TotOffenders | Total number of civilians with criminal propensity that commit a robbery | Ratio |
| TotVictims | Total number of civilians who are victims of street robbery | Ratio |
| TotRepeatVictims | Total number of civilians who are repeat victims of street robbery | Ratio |
| AveAwayTime | Average amount of time agents spend away from home | Ratio |
| | | |
| **Individual-level Process** | | |
| AwayTime | Total time spent away from home | Ratio |
| TotOff | Total robberies committed | Ratio |
| TotVict | Total times robbed | Ratio |
| Criminal Propensity | Presence or absence of criminal propensity | Dummy |
| WealthBegin | Beginning amount of wealth | Ratio |
| WealthEnd | Ending amount of wealth | Ratio |
| | | |
| **Place-level Process** | | |
| TotRobPlace | Total number of robberies | Ratio |
| TotVisits | Total number of times an agent stopped | Ratio |
| TotalNodeswRob | Total number of street nodes that had a robbery | Ratio |
| TotNodeswMultRob | Total number of street nodes that had more than one robbery | Ratio |
| MeanRobPlace | Mean robberies per street node | Ratio |
| MeanVisitsPlace | Mean visits per street node | Ratio |

## 3.3 Achieving Agent Movement in the Model

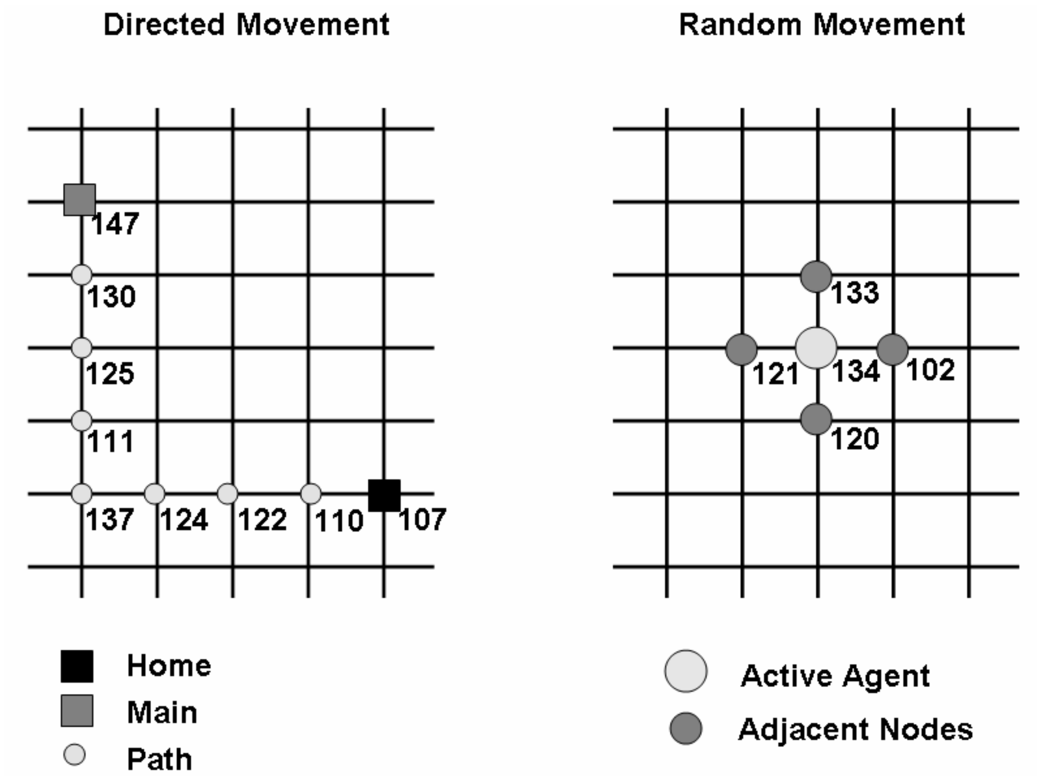
Two types of agent movement are implemented in the model, random and directed (i.e. among a predefined set of locations). These two types of movement require different strategies for implementation but both rely on street nodes rather than streets. This unconventional strategy is necessary because Agent Analyst does not support connections to a geodatabase or a network dataset, the two data structures which enable routing in ArcGIS. Consequently, there can be no dynamic routing of directed agent travel within Agent Analyst. The alternative strategy implemented here uses GIS to convert the street intersections to a node layer. An additional benefit of this strategy is that it allows dynamic agent movement to be implemented within Agent Analyst.

Directed movement in the model is the more complex type of movement and requires the definition of activity spaces for the agents before running the model. Activity spaces consist of four activity nodes and a list of path nodes. The list of path nodes describes the complete set of nodes to be traversed to visit all four activity nodes. Movement takes place from street intersection/node to a connected street intersection/node (hereafter referred to as street nodes). Since routing in ArcGIS uses the streets, identifying the street nodes that are traversed in the course of visiting all four activity nodes required the creation of a custom program.[6] The output of the program provides a list of street nodes that are traversed while traveling the shortest path among the activity nodes. Civilian agents in the model always travel among their activity nodes in the same order each day (i.e. home, main, activity one, activity

[6] The custom program was created in Visual Basic and added to the ArcGIS 9.1 session to identify the street nodes traversed by each agent. The author gratefully acknowledges the assistance of Mary Jo Fraley who wrote the code and is making it available via the author.

two). They always start and end a model day at their home. As seen in left panel of Figure 2-2, directed agent movement occurs from node to node along a pre-defined path. In the example, the agent starts at home and moves to node 107. From there the agent moves to 110, 122 and so on.

Figure 2-2: Movement in the Model



In addition to the directed movement by civilians going about their daily activities, dynamic random movement is also implemented in the model. Random movement is used by the police agents in all three versions and by the civilians in two versions of the model. Implementing random movement of civilians enables the comparison of outcomes to those from a model in which the civilians have directed movement.

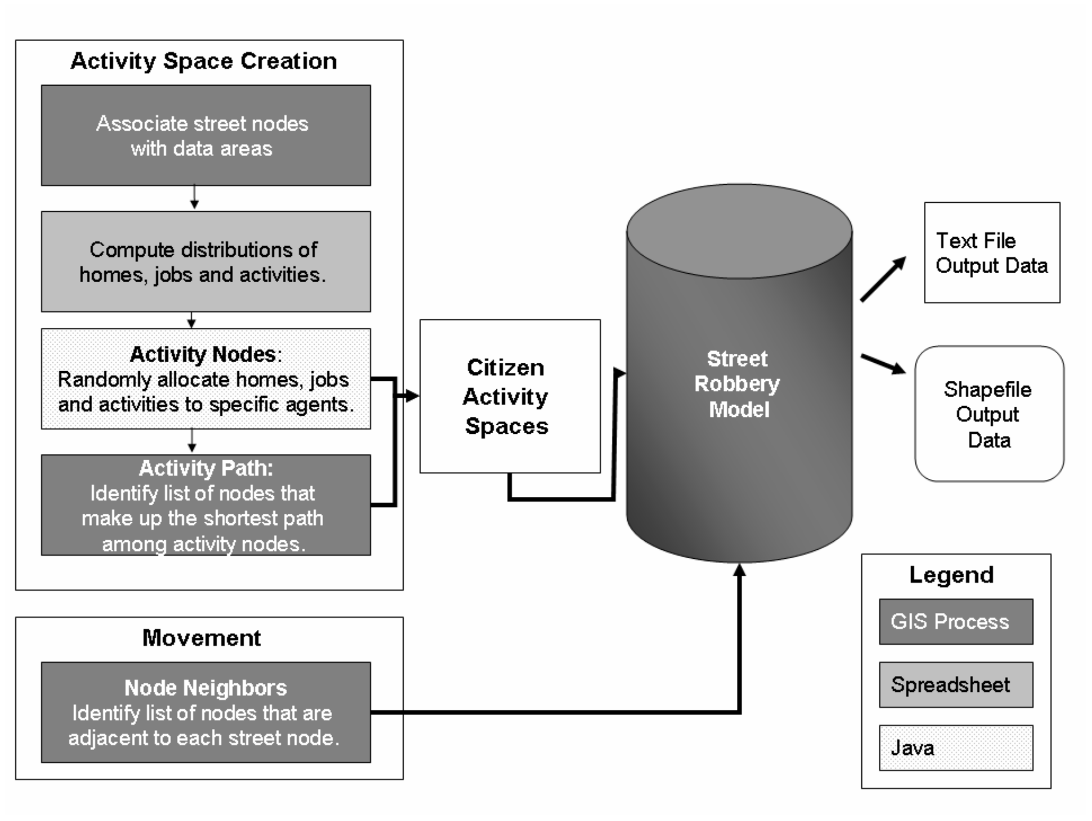
Random movement is implemented in the model through a two-step process, identification of neighboring nodes and random selection of the target node. As part of the preprocessing of data done before running the model, a set of adjacent nodes is identified for each node and written to a file. The creation of this file is achieved through a series of topological queries (i.e. select node, select streets adjacent to selected node, select nodes that intersect selected streets). In this way, a file of neighboring nodes is created for each travel node and then used as the basis for random movement.

When traveling in a random fashion, the agents follow a 'random walk' where the agents move one randomly chosen node each minute of the model (Chaitin, 1990). The current location of each agent is associated with a street node. During the model run, the file of node neighbors is used by each agent as they travel. The right panel of Figure 2-2 shows a simplified travel movement. The agent is at node 134 and could potentially move to any of the following nodes: 102, 120, 121, or 133. A uniform random number is generated giving each node an equal chance of being selected. The agent then moves to the selected node and the cycle repeats.

## 3.4 Creating Activity Spaces for Civilians in the Model

As discussed earlier, theory from both geography and criminology holds that the travel behavior of individuals is influenced by the street network, the specific locations at which opportunities for employment, recreation, retail and services exist, and the distance among those locations. The temporal schedule (i.e. the amount of time spent at each activity) is affected by the distances between the activities and the speed of travel. The more time spent traveling the less is available to spend at an activity. The complete process of developing agent activity spaces is detailed in Figure 2-3 which also describes the entire data flow from input through output.

Figure 2-3: Data Inputs/Outputs Related to Agent Activity and Outcome Data



Activity Space Creation
Associate street nodes with data areas
Compute distributions of homes, jobs and activities.
Activity Nodes: Randomly allocate homes, jobs and activities to specific agents.
Activity Path: Identify list of nodes that make up the shortest path among activity nodes.
Citizen Activity Spaces
Street Robbery Model
Text File Output Data
Shapefile Output Data
Movement
Node Neighbors Identify list of nodes that are adjacent to each street node.
Legend
GIS Process
Spreadsheet
Java

However, before the activity spaces for the civilian agents can be created the locations of the street nodes have to be linked to the polygon layers. The process begins by using GIS to create a layer of street nodes and assign area identifiers (e.g. blockgroup or zip codes) to each street node (e.g., street node 1 is in blockgroup 201). In stage 2, the distribution of homes, jobs and retail/service/recreation activities across Seattle is calculated. These distributions are then used to assign agent homes, jobs and activities in the same proportion as they are found in Seattle (e.g. if 10% of the population lives in a particular blockgroup then 10% of the agents are assigned to that blockgroup). This process produces two files; one file contains the activity node number and the blockgroup in which it is located and the other file contains the blockgroup and number of agents to be assigned a home node from that blockgroup. The same basic methodology is then repeated to assign work places and activities.

Stage three uses the two files just described in a java program that randomly selects and assigns agent homes, work places, and activities in the same proportion as they are found in Seattle. Each agent's four activity nodes are selected representing a home node, main node (e.g., work, school etc.), and two additional activity nodes (e.g., retail stores, gym, coffee shop). Two thousand files path files are written out; one for each agent when employed and another for each agent when unemployed.

The final stage in creating directed movement paths involves finding the shortest path among the nodes. As previously mentioned, the shortest path among the activity nodes is calculated using ArcGIS Network Analyst via a custom Visual Basic program that generates a list of nodes that are traversed while traveling the shortest path and writes them out to agent path files. Two paths are created for each agent; one describes their activity space when employed and the other when unemployed. Three of the four nodes remain the same between the two activity spaces, home, recreation node 1 and recreation node 2; only the main node changes. When employed, the agents' main node is assigned in the same proportion as employment; when unemployed it is assigned from the distribution of activities in Seattle. The 4,000 output files describing the activity nodes (N=2000) and activity paths (N=2000) for each agent are then ready to be used to define directed civilian agent movement in the model.

At this point, a temporal schedule is assigned within the street model using the following steps. First, the time spent at home is randomly assigned to each agent so that the societal average matches the average for the experimental condition being tested. Next the number of nodes traversed is counted and subtracted from the total time away from home. The larger the geographic extent of an individual's activity nodes the greater the time required to travel among them. The remaining time is randomly allocated to the Main, Activity 1 and Activity 2. Because of the large size of some of the agent activity spaces, the agents must travel more than one node each turn in order visit each activity node and make it back home in one day. The number of nodes traveled per turn is determined via a random normal distribution (mean = 6, sd = 1).

## *4.0 Implementation Model*

This section explains how the conceptual model of street robbery is implemented in three progressively more complex versions using ABM and GIS and based on GIS data describing Seattle (Table 2-3). The simplest version called Simple, has agents move along a real street network but does not incorporate the notion of routine activity spaces (temporal or spatio-temporal). The Temporal version has agents with random movement and a temporal schedule while in the Activity Space version agents have a spatio-temporal schedule with defined movement patterns that are based on the activity spaces developed earlier. The offender's decision making process is identical for all three versions of the model.

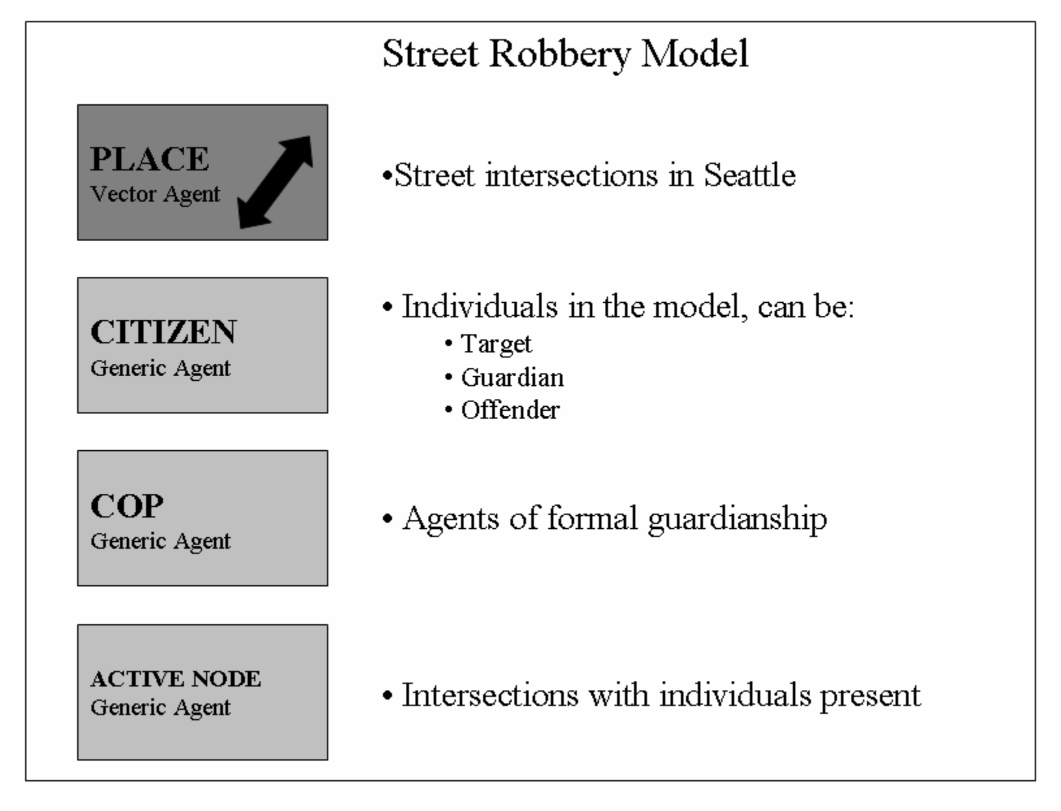
Table 2-3: Implementation Versions of the Conceptual Street Robbery Model

| | **Simple** | **Temporal** | **Activity Space** |
|---|---|---|---|
| **Civilian Movement** | Random | Random | Defined Activity Space |
| **Police Movement** | Random | Random | Random |
| **Civilian Characteristics** | | | |
| Criminal Propensity | Yes | Yes | Yes |
| Wealth | Yes | Yes | Yes |
| Activity Space | No | Temporal schedule | Spatio-temporal |
| Multi-faceted Risk Status | No | Yes | Yes |
| Employment Status | No | Yes | Yes |

### 4.1 Overview of the Landscape and the People in the Model

The model of street robbery is based on the core elements of routine activity theory (RAT): a motivated offender, a suitable target, and the lack of a capable guardian. The size, shape and timing of the routine activity spaces developed in section three are important in determining the frequency with which those elements converge in space-time. These concepts form the basis for three of the agent classes in the model: place, civilian and police officer (Figure 2-4). The fourth class consists of the active nodes and is used to keep track of the street nodes where there are agents. The rest of this section describes each of the four major components of the model: landscape, people, activity spaces, and model behavior.

Figure 2-4: Classes in the Street Robbery Model


Street Robbery Model
PLACE
Vector Agent
•Street intersections in Seattle
CITIZEN
Generic Agent
• Individuals in the model, can be:
• Target
• Guardian
• Offender
COP
Generic Agent
• Agents of formal guardianship
ACTIVE NODE
Generic Agent
• Intersections with individuals present

The city of Seattle is used as the basis for the landscape in all three versions of the street robbery model. One function of the landscape is to provide information on the distribution of population, jobs, and service/retail opportunities across Seattle. The other is to realistically structure the movement patterns of both civilians and police. Two classes in the model have to do with the landscape, places and active nodes. The place class of agents in the model represents all 16,035 street nodes. Each place has attributes that are updated during the model run (e.g. total robberies, total visits etc). Places, as the lone vector agents, are directly linked to a shapefile representing street intersections and provide the only mechanism for visualization of the model while it is running and after the completion of a model year.[7] The active node class is

[7] There are two types of agents in RepastPy GIS Model, vector and generic. Vector agents are associated with a shapefile and can thus be displayed on a map while generic agents cannot.

generic, and serves as a computational device to identify which nodes have agents on them at each tick of the model. It is dynamic and changes with each minute of the model. The active node class improves the performance of the model by restricting the set of places that have to be checked each minute of the model.

Two agent classes operationalize people in the model, civilian and police. The civilian class represents the general population of Seattle. The three roles that people can take on during a crime event are encompassed in the civilian agent class; civilians can be offenders, targets, or agents of informal guardianship. The particular role a civilian agent takes is driven by their characteristics and the contextual dynamics of the specific interaction. Police are the agents of formal guardianship. Both police and civilian agents are assigned a type of movement that is static over a particular model run. Only civilians have additional attributes that are used in the model.

Police agents have only one role, that of a formal guardian. In the model, the presence of a police agent prevents a crime from occurring. At the start of the simulation, police agents are randomly distributed across the nodes. To accomplish their mission of crime prevention police agents follow a ‘random walk’ movement pattern in which they move one node at a time and only to an adjacent node. Police never commit crimes in this model and they are never targets.

Civilian agents are randomly distributed across the nodes in the Simple and Temporal versions but in the Activity Space model civilians begin each day at an assigned home node. Regardless of version, the civilian agents in the model are assigned three characteristics that are integral to the decision to commit a street robbery: 1) a time to spend at home; 2) criminal propensity; and 3) wealth. In the

Temporal and Activity Space models agents also have an employment status. The final characteristic of agents is their mode of movement and accompanying temporal schedule which vary by version of the model; totally random, temporal only with random movement or spatio-temporal (these were discussed in section 4.0). Some background on each of the roles (i.e., offender, target and guardian) and how they are incorporated into the model is provided next.

All civilian agents are assigned a time to spend at home that is static over a model run. In the Simple model, civilians are either at home or not at home. When not at home, agents move along the street network as described in the earlier section on random movement. Civilians in the Temporal and Activity Space models share the same temporal schedule for activities and travel. They have attributes describing the time to spend at home, a main activity and two other activities. Civilians in the Activity Space model have places at which those activities occur. When not at home, civilians in the Temporal version travel randomly and those in the Activity Space version follow the shortest path among their activity nodes.

Criminal propensity is used to differentiate agents who evaluate situations and make the decision to offend from all other agents in the model. In all other ways, civilians with criminal propensity are exactly the same as those without. While only agents with criminal propensity can make the decision to offend, it is the particular constellation of individual and situational factors that determines whether a crime is committed. In this way, patterns of offending and victimization are allowed to emerge from decisions made by individuals in particular contexts.

The additional characteristic of employment status is added to civilian agents in the Temporal and Activity Space versions of the model. This characteristic has two important impacts in those versions of the model. First, it changes the relative amount of time spent at the three activity nodes (but not the overall time spent away from home). In the Activity Space version, a change in employment status also changes the activity nodes that are visited by the affected agents. Incorporating this characteristic enables the model to reflect the impact that employment status has on the temporal and spatial aspects of routine activity schedules. Second, employment status impacts the wealth of the civilian agents. Those who are employed receive regular but static infusion of wealth every two weeks over the model year. Civilians who are unemployed do not get paid. Every month, three percent of unemployed agents become employed and are replaced by a new random selection of employed agents who become unemployed. It is important to note that the employment status is assigned independently of the criminal propensity indicator; civilians with criminal propensity can be employed in the model, as they are in life.

As noted earlier, the built environment influences the structure of routine activities. The structure of routine activities, in turn, impacts the convergence of offenders, targets, and guardians at places. The interaction among civilians and police in the model takes place in the particular situational context existing at places (Carlstein & Thrift, 1978; Meier et al., 2001; R. J. Sampson, 1993). The three versions of the model implement different activity spaces for civilian agents. In the Simple model, civilian agents are assigned only a time to spend at home. All agents begin at home and then travel randomly until the end of the day. Their next day begins at the node where the previous one ended. Since they are at risk of being robbed whenever they are not at home, civilians in the random model have the highest level of risk.

The Temporal and Activity Space versions of the model incorporate a more complex notion of activity space and risk. In these versions of the model, civilian agents are not at risk when they are at home or at work; only when they are at other activities or traveling. This representation of risk is in keeping with the crime being studied. By definition, street robbery happens only on the street or in public places; not in a home or inside a workplace.

Civilian agents in the Temporal version spend the same amount of time at the four activity nodes and in travel as their corresponding civilians in the Activity Space version but they travel randomly. Only the Activity Space version uses the pre-defined activity spaces that reflect the distribution of activity places in Seattle.

The following describes general model behavior. Each tick corresponds to one minute of time, and each minute the nodes with an agent present are evaluated. Active nodes meeting the three criteria continue to be evaluated: 1) no police present, 2) at least two civilians present and 3) at least one of the civilians must have criminal propensity. First an active offender agent is selected from the agents at the active node. If there is only one offender at the node, they automatically become the active offender. Otherwise, the active offender is randomly selected from the list of agents with criminal propensity who are at the node. Random selection is necessary to ensure the same agent is not selected to be active each time the model is run. Offender agents who are not selected are at risk of becoming victims. Once the active offender at each of the active nodes evaluates their situation, all agents move and the decision structure repeats.

## *5.0 Results of Implementing a Model in Agent Analyst*

The successful implementation of a theoretically-based, geographically-aware model of street robbery documented here capitalizes on the recent development of Agent Analyst to enable agent movement along real street networks. It also demonstrates how realistic activity spaces can be developed from the distribution of land use in a city. These advances allow a simulation model to be 'situated' on an existent rather than an artificial landscape and in doing so provide a more realistic context to the model.

The versions of the model implement two types of movement along a street network, random and directed. Both implementations 'situate' simulated interactions by structuring potential movement based on a street network. The random movement offers the ability to have agents move along an existing street network rather than in abstract grid space or along linked grids to mimic a street network. Thus researchers can use a street network directly without having to convert the network to a grid for use in a simulation model.

Although this is a significant advance, there are two drawbacks to the current implementation of random movement. First, the random selection of the next node is done from a list that only considers adjacent nodes. Thus, agents are limited to moving only one node per minute. In addition, there is no prohibition against backtracking by agents. The same node that an agent just came from is included in the list of adjacent nodes for the current node which means it can be selected as the goal node. Together these implementation decisions may lead to smaller, less realistic activity spaces for agents that are moving randomly. Future implementations should consider giving agents who are moving randomly, the same ability to move more than one node as the agents who have directed movement.

The directed movement allows agents to move purposefully among a set of locations. These locations can represent home, work, school, shopping, service, and/or recreation places. Directed movement serves as the basis for spatial activity spaces by enabling agent travel among a set of predetermined locations that are visited daily.

The creation of activity spaces also is an important step forward in situating simulation. The model versions implemented here will provide the basis for systematic experiments testing how a random schedule of activity, can be compared to one with temporal constraints and one with spatio-temporal constraints. Thus the impact of time can be tested separately from the impact of space. Although this model implements only simplified versions of activity spaces, the methodology provides a guide for future research to extend.

## *6.0 Conclusions*

This paper details a new methodology for 'situating' simulation. It makes use of a recently released software tool that integrates GIS and ABM. The result is a package that combines the individual-level modeling capabilities of an ABM with the spatial analysis capabilities of a GIS and in doing so situates simulated agents within an empirical context. A methodology for enabling agent movement on a street network and creating geographically informed activity spaces is detailed and provides the foundation for the development of three versions of a model of street robbery events.

This implementation of a street robbery model has important implications for the use of simulation to elaborate theory (Albrecht, 2005; J. Eck, 2005) and to conduct experiments (Dowling, 1999; Schultz & Sullivan, 1972). Previous attempts to test routine activity theory, although generally supportive, have produced mixed results (Kennedy & Forde, 1990; Messner & Blau, 1987; Miethe & McDowall, 1993; R. Sampson, J. & Lauritsen, 1990). None of those tests were able to sufficiently address the spatio-temporal structure of routine activities, satisfactorily deal with measurement issues or effectively capture the dynamic nature of interactions at the micro level. The methodology and model implemented here address all three of those issues. In addition, agent based software allows single aspects of the behaviors of agents to be manipulated while all others are held constant. This capability is used to conduct controlled experiments to test the core axioms of routine activity theory (Groff, Forthcoming-a) and to separate the effects of space and time on the convergence of the elements necessary for a street robbery to occur (Groff, Manuscript available from author).

The choice of parameter values is a critical aspect of all models that deserves special attention because it impacts the external validity of the model by compromising the parameter validity; the measure of how well the parameter values used in the model matched reality (Carley, 1996). As noted in the data section, this research made every attempt to use realistic model parameter values. However, in some cases there was no evidence available and in others a simplified representation was chosen to establish a baseline (e.g. wealth distribution) (Axelrod, Forthcoming; Epstein & Axtell, 1996). In these cases, the validity of the parameters is unconfirmed and their impact on the model results needs to be thoroughly tested via systematic sensitivity tests.

On the whole, the methodology presented here is relatively straightforward and establishes a foundation for further more complex explorations of agent movement and activity spaces. In the case of agent movement, one enhancement would be to take into account barriers. For example, in the case of police, they typically are assigned to patrol within designated areas. Subsequent implementations could make use of those types of barriers and in doing so provide a step toward more realistic police agent behavior. Barriers are also important to civilians and may be physical (e.g. streams, limited access highways) and/or perceptual (e.g. edges of neighborhoods etc.). Another enhancement would be to use speed limits and one-way streets in the development of the routes among activity places.

The future use of this methodology is both facilitated and limited by the software packages available for implementation. Currently, Agent Analyst offers a unique opportunity and some challenges to 'situating simulation'. Since it is still under development, its ultimate form is still evolving so all these statements apply to the beta version only. On the plus side, Agent Analyst offers the most straightforward option for non-programmers who are interested in developing their own spatially-aware models. It unburdens the new programmer from the numerous details involved in developing the model framework and learning the syntax of java. On the minus side, the current version is only able to read shapefiles, not the network data sets that would enable dynamic routing. In addition, the debugging tools are extremely limited. Finally, would-be modelers must become familiar with a unique subset of Python syntax and any Java classes that are used. Addressing these issues would decrease the difficulty of programming models, speed development time, and increase the realism of agent travel and activity spaces in the models.

# Chapter 3 Simulation for Theory Testing and Experimentation

## *1.0 Introduction*

Achieving a better understanding of crime events in their spatio-temporal context is an important research area in criminology with major implications for making better policies and developing effective crime prevention strategies. Theoretical advances under the rubric of opportunity theory have highlighted benefits of a shift in focus from the criminal motivation of people to the contexts in which crime events occur (Paul Brantingham & Brantingham, 1981a; J. E. Eck & Weisburd, 1995; D. L. Weisburd, 2002). Because these approaches focus on the crime event and not the intrinsic motivations of the actors, they produce concrete and immediate strategies for both policy and practice (Akers, 2000; Cullen & Agnew, 1999; Felson, 1987; Vold et al., 2002). Implementation of these strategies holds the promise of quick and measurable reductions in crime rates.

Routine activity theory (RAT) (Cohen & Felson, 1979), in particular, has received a great deal of attention and its crime reduction potential is widely recognized.[8] Accordingly, there have been many attempts over the last twenty-five years to empirically validate routine activity theory. Despite applying a variety of methodologies, these studies have produced inconsistent support for the theory. Their

[8] Environmental criminology is another important theory that emphasizes place characteristics and offender travel in the convergence of victims and offenders in space-time (Paul Brantingham & Brantingham, 1991 [1981]; P. J. Brantingham & Brantingham, 1978). Other theories relevant to micro level modeling include lifestyle theory (Hindelang, Gottfredson, & Garofalo, 1978) and the criminal event perspective (CEP) (Meier et al., 2001). However, the need to focus on one theory for the initial model precludes a full examination of these theories. Three books on the theoretical foundations of criminology offer a more complete overview of opportunity theories (Akers, 2000; Cullen & Agnew, 1999; Vold et al., 2002).

shortcomings stem from the lack of: 1) individual-level data, and 2) flexible modeling tools.

This research demonstrates a new approach to testing routine activity theory (Cohen & Felson, 1979) using simulation modeling. The assumptions of routine activity theory are operationalized and implemented in a model of street robbery events. The crime of street robbery is a natural choice for this type of model because it involves the interaction of individuals in a public area (e.g. street, parking lot, recreational area etc.). In addition, it is an instrumental crime (for economic gain), and thus more likely to be the result of a rational decision than an expressive crime (e.g. assault) (Clarke & Cornish, 1985; Walsh, 1986).

The model findings provide strong evidence for the plausibility of routine activity theory's core proposition; as individuals spend more time away from home, the number of street robberies will increase. Analysis of the spatial patterns of street robberies reveals a clustered distribution that becomes more dispersed as time away from home increases since individuals have more time to travel farther from home. In addition, locations that have a high incidence of robbery when society, as a whole, is spending less time away from home remain high density locations as society spends more time away from home.

The following sections provide the rationale and methodology for constructing simulation models based on theory. Section 2 discusses previous tests of routine activity theory and provides an introduction to simulation as an alternative methodology. The criminological concepts that underpin the model are identified in section 3. Although routine activity theory is the focus of this research, rational choice theory contributes concepts important to the rules that guide offender decision-making. Section 4 provides a description of how those constructs are implemented in an agent-based modeling (ABM)/geographic information systems (GIS) framework. The analysis strategy and findings are detailed in Sections 5 and 6. The final sections discuss the findings and implications for future research.

## *2.0 Meeting the Challenges Encountered by Previous Research*

A wide variety of studies have attempted to validate routine activity theory. Some have employed macro-level data to approximate the construct of routine activities (Cohen, 1981; Messner & Blau, 1987; Miethe, Hughes, & McDowall, 1991). Others have relied on survey data collected from individuals (Miethe, Stafford, & Long, 1987; Osgood, Wilson, O'Malley, Bachman, & Johnston, 1996), and still others have combined micro- and macro-level variables to represent routine activities within a social structure (Cohen, Kluegel, & Land, 1981; Kennedy & Forde, 1990; Miethe & McDowall, 1993; Rountree & Land, 1996; R. Sampson, J. & Lauritsen, 1990; R. J. Sampson & Wooldredge, 1987). As mentioned earlier, these studies have found inconsistent support for the theory. However, the studies suffer from three main shortcomings: 1) failure to consider the spatio-temporal structure of routine activities, 2) measurement issues, and 3) the inability to represent patterns emerging from individual-level interactions. These issues are addressed in more detail next.

Although the importance of spatio-temporal elements in routine activities is often acknowledged, the spatial structure and timing of these activities has been widely overlooked.[9] Indeed none of these studies incorporated the dynamic, spatio-temporal interaction of offenders, victims, and potential guardians at the micro level; an omission that was most likely driven by a lack of data. In a commendable effort, two studies attempted to address these issues through the inclusion of gross measures to capture the timing of routine activity (e.g., breaking out daytime from nighttime activities) (Kennedy & Forde, 1990; Miethe & McDowall, 1993). However, the spatio-temporal structure of routine activities is a core component of routine activity theory and must be more comprehensively measured if its role in the convergence of offenders and targets is to be better understood.

A variety of measurement issues arise when attempting to test routine activity theory (Cohen et al., 1981; Miethe et al., 1991; R. J. Sampson & Wooldredge, 1987). As Bursik and Grasmick note "it has been notoriously difficult to collect reliable and valid indicators of its central components" (1993 77). Other measurement issues include: ecological fallacy; overlapping operationalization of constructs; difficulty with adequately measuring the construct of routine activities; and a reliance on official data and victimization surveys that have widely-recognized flaws. When tests are done using macro-level data, they are susceptible to the ecological fallacy which states that the characteristics of an area cannot necessarily be inferred to individuals. Consequently, macro-level data are generally unsuitable for testing a micro-level theory such as routine activity (J. E. Eck, 1995a).

Regardless of the level of analysis, all studies have struggled with measuring the construct of routine activities as isolated from other constructs being measured. This

[9] Two studies (Miethe & McDowall, 1993; R. J. Sampson & Wooldredge, 1987) emphasized how opportunity structure changed across areas but neither measured how the spatio-temporal structure of routine activities impacted the observed distribution of crime.

problem is related to general issues that have arisen when attempting to clearly link empirically measured variables to particular constructs (e.g., single person households are associated with less informal social control and with less guardianship) (Cohen et al., 1981; Miethe et al., 1991). These issues make it difficult to test the theory because data issues rather than theoretical ones can be employed to dispute contrary evidence (Miethe et al., 1991; Miethe et al., 1987). In addition, the reliance on official data and victimization surveys, which have widely-recognized flaws, makes conclusions drawn from studies using those sources susceptible to the usual caveats (Gove, Hughes, & Geerken, 1985).

Finally, all of the previous tests reviewed here suffer from the inability to adequately model the complex and dynamic interactions of individuals that produce observed crime patterns. Routine activity theory is essentially a micro-level theory with macro-level implications; it characterizes crime patterns as resulting from the decisions of individuals made in the context of a particular situation (J. E. Eck, 1995a). The methods of previous studies were simply not able to accommodate the complex, non-linear nature of constantly changing individual-level interactions and the manner in which crime patterns emerge from those interactions (Liu et al., 2005).

## 2.1 A New Approach for Modeling Crime Events and Crime Patterns

Simulation modeling offers an alternative method for capturing the dynamic interactions among individuals taking place at the micro level and their relationship to macro level patterns. Some researchers view simulation as a third way of conducting social science research in addition to the more traditional verbal and mathematical/statistical representation of theories (Gilbert & Terna, 1999; Ostrom, 1988). In this tradition, simulation allows for the exploration and elaboration of theory (Dowling, 1999; J. Eck, 2005; O'Sullivan, 2004a). Like other modeling approaches, simulation modeling involves the creation of a simplified representation of a social phenomenon (Gilbert & Terna, 1999). The most familiar type of model is a statistical one (e.g., a regression model) in which input data are 'run' via a statistical program and values are output that describe the relationships among the input data. In contrast, simulation models are themselves computer programs that incorporate the critical aspects of the social phenomenon being modeled. The program is run and the output data are analyzed, often via standard statistical techniques. Simulation modeling has two main advantages over statistical models. It allows heterogeneity among individuals that more closely approximates the variety found in everyday life and is able to accommodate the non-linear relationships present in dynamic and complex interactions (Dibble, 2003; Epstein & Axtell, 1996; Gilbert & Terna, 1999).

Agent-based modeling (ABM) is one type of simulation. ABM employs a bottom-up approach; agents are imbued with unique characteristics and general behavioral rules and macro-level patterns emerge from their interactions (Epstein & Axtell, 1996; Gilbert & Troitzsch, 1999). An agent "can be thought of as an autonomous, goal-directed software entity" (O'Sullivan & Haklay, 2000 13). Agents most often represent people but can also correspond to organizations, neighborhoods, governments etc. The characteristics of agents can be randomly assigned so that specific societal averages are produced and the possibility of systematic bias is all but eliminated. Individual agents in the model interact with each other based on a set of decision rules. Their characteristics are dynamically changed as a result of those interactions. Traditionally, agents interact in an artificial world, although the value of leveraging GIS data to provide a 'real' landscape is gaining recognition since artificial landscapes do not take into account the impact of the environment on agent behavior (Brown et al., 2005; O'Sullivan & Haklay, 2000).

Additional scientific rigor is achieved when simulation models are implemented within a computational laboratory framework (Dibble, 2003;2006; Epstein & Axtell, 1996; Gilbert & Terna, 1999; Macy & Willer, 2002).[10] Computational laboratories enable experiments to be conducted and replicated. Aspects of the agents, society, and the landscape can be held constant or systematically varied in order to provide a level of control impossible to attain using traditional social science methods. These characteristics of computational laboratories facilitate the creation of a variety of simulated experiments, featuring different conditions or applying various prevention scenarios, which are then evaluated. An added advantage is that compared to empirical research, simulations have minimal cost.

Recently, a small body of research has emerged that makes use of simulation models to explore crime-related issues. Work by Epstein, Steinbruner and Parker (2001) on civil violence and Wilhite (2001) on protection and social order provide interesting approaches to modeling how the interactions of individual agents are related to emerging patterns of violence or protection. Within criminology, work has begun on conceptualizing the application of simulation in environmental criminology (P. L. Brantingham & Brantingham, 2004; P. L. Brantingham & Groff, 2004) and explaining crime patterns (J. Eck, 2005). ABM is being applied to study both

[10] The term *computational laboratory* refers to the software tools to create and evaluate models through systematic experimentation and descriptive analysis of output data (Dibble, 2003;2006; Epstein & Axtell, 1996; Gilbert & Terna, 1999).

physical and cyber crime (Gunderson & Brown, 2003), and some researchers are combining ABM with other technologies. One example implements a general model of crime on a GIS-based raster grid (Wang et al., 2004). Another study, based on routine activity theory, employs cellular automata to study street robbery in one neighborhood (Liu et al., 2005). Rather than offering competing paradigms, these approaches represent a healthy variety of complementary approaches (J. E. Eck & Liu, 2004).

The approach taken in this paper extends previous efforts in several ways. First, the steps involved in building and applying a simulation model are thoroughly explained to aid in replication. Second, a set of experiments is conducted to provide the first direct test, albeit in an artificial society, of Cohen and Felson's core assertion that shifts in routine activities away from home, increases crime rates. Each experiment holds the number of motivated offenders and suitable targets constant, only the amount of time spent at home varies for the agents in the model. Third, software integrating ABM and GIS is used to explore how agent travel on a real street network impacts the frequency of convergence of the elements necessary for a crime to occur. GIS software excels at managing data about space and ABM is superior at keeping track of time; together they allow exploration of space-time relationships. Finally, the new approach allows examination of how the convergence of heterogeneous agents translates into aggregate rates of street robbery.

The remainder of the paper details how a computational laboratory using ABM/GIS can be employed to address the following research questions: 1) Does the shift of routine activities away from home increase the incidence of street robberies?;

and 2) What is the impact of increasing time spent away from home on the spatial pattern of street robberies? In order to facilitate interpretation, the initial model is made as simple as possible implementing only the core concepts of routine activity.

## *3.0 Theoretical Basis for the Conceptual Model and Behavioral Rules*

Although the approach advocated here is novel, the process of developing models to represent reality is not. Models have a long history of use in the general social sciences (Gilbert & Terna, 1999; Gilbert & Troitzsch, 1999; Schelling, 1971; Simon, 1952). While models vary in how faithfully they represent reality, they typically operate on the principle that simpler is better; thus a primary goal of modelers is try to assemble the most parsimonious model to answer a question. The degree to which the theory is represented in the model represents the structural validity of the model (An et al., 2005; Manson, 2001). Simulation models, in particular, start with simple models and then systematically add complexity to ensure that the dynamics are well understood before continuing (Macy & Willer, 2002).

Following those earlier modeling efforts, the building of the simulation model detailed here begins with the identification of the most basic theoretical propositions of routine activity theory. Once these are identified, the next step is to develop a conceptual diagram that captures both the essential constructs and how they are related to one another. The constructs and their relationships are then formalized so they can be coded in a computer program. In some cases, the constructs are formalized as clearly stated verbal guidelines that underlie the behavior of agents, their interactions with other agents, and their interaction with the environment. In other cases, the definition of these constructs takes the form of mathematical equations for evaluation of specific situations an agent encounters during the course of a simulation. Where theory is not detailed enough for implementation or does not address an issue, empirical research is used to enhance the representation of behavior within the model. The final step in building the simulation is implementation of the model via a software package that integrates ABM and GIS.

Fortunately, there is rich background literature to guide the development of an agent-based model of street robbery. Appropriately, the model relies mainly on routine activity theory for definition of the core concepts. Since routine activity theory does not address offender decision-making, rational choice perspective is employed to develop the decision rules applied in the model (Clarke & Cornish, 1985;2001). The next sections serve a dual purpose providing the basis for both the conceptual model and the formalization of behavioral rules.

## 3.1 Routine Activity Theory

Cohen and Felson's (1979) original formulation of routine activity theory has become the most frequently cited basis for examining the connection between social structural changes that have affected routine activities and crime.[11] In their seminal work, Cohen and Felson hypothesize that it was the shift away from home-based activities that produced the increase in crime that occurred after World War II. An increase which occurred despite improvement in the socioeconomic indicators historically associated with crime (e.g. unemployment and education). As originally conceptualized, routine activity theory identifies the convergence of *motivated offender*, *suitable target*, and the *lack of a capable guardian* at a particular place and

[11] The extensions to the original 1979 version of the theory are not incorporated into this first effort (Felson, 2001;2002). This was done in order to make the results of the model easier to interpret.

time as the core elements necessary for a crime to occur (Cohen and Felson 1979). The authors also recognize the importance of *routine activities* in shaping the spatio-temporal structure of convergence of victim and offender. They emphasize that crimes occur when the normal everyday activities of offenders and victims converge with no guardian present.

The four elements of a crime noted above form the main constructs of the model (Figure 3-1). There are two types of people in the model, civilians and police. Civilians take on roles representing the three major elements of crime (i.e. offenders, targets and guardians). The fourth element, routine activity, is influenced by the amount of time an individual spends away from home and the network of streets available for travel. Once convergence occurs, factors such as guardianship and suitability of target are considered by the offender when making the decision whether or not to commit a robbery.

Figure 3-1: Simple Conceptual Model of Street Robbery

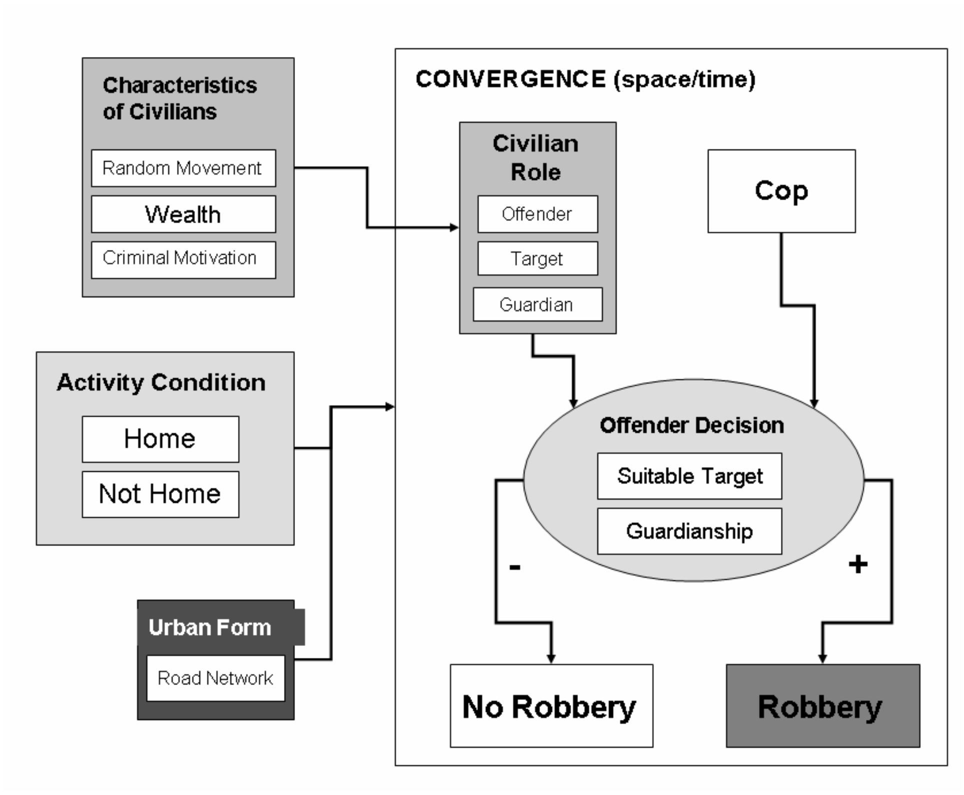


Table 3-1 and the next few paragraphs provide a detailed account of how each of the above constructs is translated into a formal verbal description and then how that description is implemented in the model. Beginning with motivated offenders, RAT assumes they are ubiquitous. In making the decision to offend, they evaluate the level of guardianship and whether or not a suitable target is present. Targets, on the other hand, are central to the theory which identifies visibility, accessibility, ability to self-protect, and potential for financial gain as the most important characteristics in determining their suitability. The specifics of what constitutes a capable guardianship are not addressed, but the theory suggests that the deterrent value of some types of individuals is higher than other types. For example, formal guardians such as police officers have greater influence because they are more likely to intervene.

Cohen and Felson view the construct of routine activity as the key *dynamic* element in determining aggregate crime rates because they affect the particular configuration of offenders, guardians and targets in a situation. Changes in routine activities directly impact the frequency of convergence among these elements which in turn, increases or decreases overall street robbery rates resulting from 'direct-contact predatory violations'.[12] In addition, the theorists postulate that if the frequency of convergence increases, crime may increase even if the absolute number of motivated offenders remains constant. The central premise of routine activity theory then is that as individuals spend more time away from home, crime will increase.

[12] Following Glaser, they define 'direct-contact predatory violations' as crimes where "someone definitely and intentionally takes or damages the person or property of another" (1974 4).

Table 3-1: Formalization of Theoretical Concepts

| Theory | Theoretical Concept | Verbal Operationalization of Theoretical Concept | Implementation |
|---|---|---|---|
| | **Motivated Offender** | | |
| -Routine Activity<br>-Rational Choice | There is a supply of motivated offenders. | Research indicates that approximately 20% of the population has participated in crime.[13] This is the proportion of the population that is encompassed in the idea of motivated offenders. They have already achieved the state of 'readiness' to commit a crime. | -Twenty percent of agents have criminal propensity.<br><br>-Only agents with criminal propensity make decision to offend. |
| -Rational Choice | Offender makes decision to offend using bounded rationality and based on the availability of suitable targets without capable guardians. | Among those individuals with some level of criminal motivation, the decision to offend utilizes information on the suitability of targets and the level of guardianship to evaluate the potential for a successful crime. The decision itself is not necessarily lengthy or rational but rather based on a form of 'bounded rationality' in which offenders choose the first opportunity that is convenient and meets some minimum requirement for risk and reward. | -Agents with criminal propensity compare wealth of other agents at node with own wealth. |
| | **Suitable Target** | | |
| -Routine Activity<br>-Rational Choice | Suitable target is an individual who is visible, accessible and has perceived value. | Visibility and accessibility requirements are met if an individual is on the street as opposed to at home or inside a building. The individual also must be perceived as having at least as much money as the motivated offender so there is some potential for gain. | -Agents with criminal propensity evaluate all other agents at the same node based on the formula below:<br><br>$S = (W_T) - (W_A) + P_S$ |

[13] Two studies that examined total lifetime participation rates for serious crimes are averaged to get the propensity applied in the model. McCord (1979) found a participation rate of 16.9% for males and Blumstein and Gaddy (1982) found a rate of 22.8% among males and females. An average of those two is 19.8% which is rounded to the 20% figure in the model.

| Theory | Theoretical Concept | Verbal Operationalization of Theoretical Concept | Implementation |
|---|---|---|---|
| | **Capable Guardian** | | |
| -Routine Activity<br>-Rational Choice | Formal and informal guardians factor into level of guardianship. | Other individuals at the same place affect the decision to offend. Their deterrent effect depends on the offender's perception that they might intervene in the crime. Police have a high deterrent effect because they are the most likely to intervene. | -Each agent with criminal propensity evaluates the level of guardianship at a node.<br><br>• If cop present, no crime.<br>• Use formula to evaluate informal guardianship:<br><br>$G = ((N_A - 2) + P_G)$ |
| | **Routine Activities** | | |
| -Routine Activity | Social structure | Changes in the social structure over impacted the amount of time spent away from home. As the locus of leisure and work time moved away from the household, fewer individuals were left in the home to act as guardians and their individual exposure to crime increased. | Average time spent away from home for all agents is systematically varied across five experimental conditions. |

### 3.2 Offender Decision-making

Since routine activity theory assumes a supply of motivated offenders and does not address the decision to offend, the rational choice perspective supplies the theoretical basis for offender decision-making in the model.[14] Rational choice defines criminal behavior as a two-step process. The first step involves the decision to participate in criminal acts. The result of this step is a state of "readiness" to commit crime. The second step involves the decision to commit a particular crime and is influenced by the situational factors that exist in a particular context. This research focuses on the second step in the process, the decision to commit a specific crime and assigns agents in the model a criminal propensity indicator that signifies they are at this stage.

There are several components of rational choice that inform the model. At the core of the theory is the concept of rationality in decision-making. The theorists advocate for a notion of rationality that is very broad, stating that "even if the choices made or the decision processes themselves are not optimal ones, they may make sense to the offender and represent his best efforts at optimizing outcomes" (Clarke & Cornish, 1985 163). In other words, offenders rely on a form of 'bounded rationality' when making the decision to commit a specific offense.[15] Rational choice perspective also assumes that, except for commission of crimes, offender routine behavior is essentially similar to that of non-offenders.

[14] Following Epstein and Axtel (1996: 1) this research is not concerned with the specific study of how individuals make decisions, the topic of studies in experimental economics that focus on game theory and decision rules, but rather examines the effect of specific individual behaviors on macro level social patterns.

[15] Bounded rationality, in particular, lends itself to investigation via agent-based models (O'Sullivan & Haklay, 2000).

The theorists also emphasize the value of models and particularly of formalizing process in models. Clarke and Cornish (1985 149) specifically acknowledge the heuristic value of models that incorporate the role of situational aspects of criminal behavior by stating "[t]hey do not have to be 'complete' explanations of criminal conduct, but only ones 'good enough' to suggest new directions for empirical enquiry or crime control policy". In addition, Clarke and Cornish suggest models should be crime specific and include situational factors (1985; 2001); thus, this research focuses on street robbery rather than robbery in general and includes both individual and situational factors.

## *4.0 Implementing a Model of Street Robbery*

The following section details how the theoretically based rules and relationships discussed earlier are implemented in a basic model of street robbery.[16] This stage of model building highlights the attributes that make agent-based simulation a powerful tool for exploring spatio-temporal behavior. Specifically, the ability to represent a group of agents as autonomous decision-makers that are involved in dynamic interactions is essential to modeling complex systems of individuals such as those producing crime. The decisions made by agents at one point in time affect the information considered by agents in subsequent turns. For example, when agents are robbed they have less money; the next time they are in a situation with a motivated offender they may be perceived as a less suitable target. This change occurs as the program is running, and its impact is immediately incorporated into the scenario. The

[16] The technical details of the implementation are available in Groff (Forthcoming-b).

outcome of the model is a set of measurements describing individual and societal crime rates and the distribution of crime events.

## 4.1 Software

The model is built using Agent Analyst software which combines two of the most popular packages for ABM and GIS. The Recursive Porous Agent Simulation Toolkit for Python (RepastPy) provides the ABM capabilities which are integrated with ArcGIS. After Agent Analyst is added into ArcGIS as a toolbox, the program can read from and write to shapefiles. Shapefiles are the program's native file format for geographic information.

## 4.2 Study Area, Duration and Data

Although the model can be implemented with any street network, the initial implementation is situated in Seattle, Washington which provides the data for the model landscape. Seattle is the largest city in the state of Washington with a population of 564,945 persons in 2000 (U.S. Census Bureau, 2000) and two-thirds of the city is bounded by water. Two types of data about Seattle are important to the research: 1) input data consisting of a GIS layer that describes the Seattle street network; and 2) output data collected during the model runs which quantify the model outcomes.

Since this is a simulation model, data representing the state of society and the state of individual agents can be produced at custom intervals (e.g. daily, monthly, etc). Data are collected over the course of one year (525,601 minutes) which allows the exploration of changes that might be occurring in the behavior of individual civilians and society over time.

Input data for the model consists of the street network of Seattle which is derived from the King County Street Network Database (SND) file and provides the basis for agent movement in the model. Because of software limitations, this layer is converted to a set of nodes that represent the street intersections. Instead of traveling along streets, civilians and police in the model move from street intersection/node to street intersection/node (hereafter referred to as node). There are 16,035 nodes in Seattle, and these locations represent places at which a street robbery may occur.

In a simulation model, the modeler controls the type of data that are collected and how frequently they are written to a file. For this research, the outcome data are collected about individual civilians, nodes (places) and society at daily intervals during the model and at the completion of each model run (Table 3-2). These data are written to two types of files, text files and shapefiles. CrimeStat and ArcGIS are used to analyze and visualize the results.

Agent level variables are collected to describe the time spent away from home, victimization, offending, wealth levels, and whether or not the civilian has criminal propensity. Cumulative totals of the results of agent interactions are also collected for society as a whole. These variables describe: the frequency of street robbery; the number of times more than one agent is at a node; the extent to which police deter crime; offending rates among civilians with criminal propensity; and victimization rates for all civilians. In addition, data from the simulation are collected at the street node level. These data are subsequently employed to generate statistics about the spatial distribution of street robberies.

Table 3-2: Outcome Data from Model

| **Variable Name** | **Description** | **Level of Measurement** |
|---|---|---|
| **Societal-level Outcome** | | |
| TotRob | Total number of robberies | Ratio |
| RobRate | Average number of robberies per population | Ratio |
| TotConverge | Total number of convergences (i.e. situations with a motivated offender and one or more 'at risk' civilians) | Ratio |
| TotDeterred | Total number of robberies deterred by a police's presence | Ratio |
| TotOffenders | Total number of civilians with criminal propensity that commit a robbery | Ratio |
| TotVictims | Total number of civilians who are victims of street robbery | Ratio |
| TotRepeatVictims | Total number of civilians who are repeat victims of street robbery | Ratio |
| AveAwayTime | Average amount of time agents spend away from home | Ratio |
| | | |
| **Individual-level Process** | | |
| AwayTime | Total time spent away from home | Ratio |
| TotOff | Total robberies committed | Ratio |
| TotVict | Total times robbed | Ratio |
| Criminal Propensity | Presence or absence of criminal propensity | Dummy |
| WealthBegin | Beginning amount of wealth | Ratio |
| WealthEnd | Ending amount of wealth | Ratio |
| | | |
| **Place-level Process** | | |
| TotRobPlace | Total number of robberies | Ratio |
| TotVisits | Total number of times an agent stopped | Ratio |
| TotalNodeswRob | Total number of street nodes that had a robbery | Ratio |
| TotNodeswMultRob | Total number of street nodes that had more than one robbery | Ratio |
| MeanRobPlace | Mean robberies per street node | Ratio |
| MeanVisitsPlace | Mean visits per street node | Ratio |

## 4.3 Hypotheses and Experiments

Two hypotheses are tested via five theoretically-based experimental conditions (Table 3-3). Each experimental condition represents an increase in the societal average for time spent on routine activities away from the home (i.e. 30%, 40%, 50%, 60% and 70%).[17] The two hypotheses examined are:

> $H_1$: As the average time spent by civilians on activities away from home increases, the aggregate rate of street robbery will increase.
>
> $H_2$: As the average time spent by civilians on activities away from home increases, the spatial pattern of street robberies will change.

The first hypothesis tests the core assertion of routine activity theory – crime rates will increase as time spent away from home increases. The second hypothesis explicitly examines the spatial structure of street robbery locations by comparing the spatial pattern produced under each of five experimental conditions.

Table 3-3: Experimental Conditions

| **Movement Type** | **Average Time Spent Away From Home** | | | | |
|---|---|---|---|---|---|
| | **C1** | **C2** | **C3** | **C4** | **C5** |
| **Random** | 30% | 40% | 50% | 60% | 70% |
| Hours per week | ≈50 | ≈67 | ≈84 | ≈101 | ≈118 |

## 4.4 Parameters in the Model

In addition to the input data describing Seattle, twelve exogenous parameters are set prior to the model run.[18] Table 3-4 describes and provides the rationale for each

[17] Using data from 1966, Cohen and Felson calculated the average time spent away from home to be 7.74 hours per day. The experimental conditions begin at 7.2 hours per day spent away from home (30 percent condition) and increase by 10 percent with each subsequent condition to a high of 16.8 hours per day (70 percent condition).

[18] The values of several of these parameters are assigned using random number generators (RNGs). In simulation models, random numbers have two main functions: 1) provide a stochastic element into what would otherwise be deterministic models of human behavior and 2) enable the replication of model results through assignment of a random number seed at the start of a simulation. The seed is the starting point for all random numbers that are produced during the course of a model run. A particular

of the parameters values in the model. The choice of parameter values is a critical aspect of all models that deserves special attention because of the potential impacts on the model outcomes. Parameterization of simulation models, while often based on empirical data, must sometimes rely on the experience of the researcher (Liu et al., 2005). For this study, every attempt is made to assign realistic model parameter values, but in cases where there was no evidence available a simplified representation was chosen to establish a baseline (e.g. wealth distribution) (Axelrod, Forthcoming; Epstein & Axtell, 1996).

seed produces the same sequence of numbers each time. This attribute enables testing of the robustness of model outcomes since in simulation modeling the results of a single model run are vulnerable to being atypical (Axelrod, Forthcoming). This research applies an explicit random number seed based on the Mersenne Twister algorithm, currently considered to be the most robust available, as the basis for all random number distributions used in the model (Ropella et al., 2002).

Table 3-4: Parameters in the Model

| Variable | Rationale |
|---|---|
| **Society Level** | |
| Number of Agents = 1,000 | Represents a balance between ensuring there are enough agents so that interactions can occur and the computational overhead from using more agents |
| Number of Police = 200 | Chosen to ensure that police would be present at some of the convergences that occur across the 16,035 places in Seattle. |
| Unemployment Rate = 6% | The unemployment rate of six percent is based on the 2002 unemployment rate for Seattle (Bureau of Labor Statistics, 2003).[19] |
| Rate of Criminal Propensity = 20% | Given that 20% of the population has committed a crime, 20% of civilians are assigned criminal propensity using a uniform distribution (Visher & Roth, 1986). |
| Time To ReOffend = 60 | Parameter value chosen as a starting point since the author could find no empirical data on which to base time to reoffend.. |
| Random Number Seed = 100 (seed also tested at 200, 300, 400 and 500) | An explicit random number seed based on the Mersenne Twister (MT) algorithm is used as the basis for all random number distributions used in the model. MT is currently considered to be the most robust in the industry (Ropella et al., 2002). |
| **Agent Level** | |
| Societal Time Spent Away From Home = 30% (40%, 50%, 60%, 70%) | Assigned based on a normal distribution with a mean of 432 minutes (for the 30% condition) and a standard deviation of 10% of the mean (sd = 43). |
| Initial Wealth = 50 | Initial wealth is assigned with a mean of 50 and a standard deviation of 20 units. |
| Amount of wealth received each payday = 5 | No empirical evidence available. |
| Amount of wealth exchanged during robbery= 1 | No empirical evidence available.[20] |
| **Situation Level** | |
| Guardianship Perception = U(-2,2) | The guardianship perception value can add or subtract zero, one or two guardians from the actual number present. This represents the stochastic element in the offender's perception of the willingness of a guardian to intervene. |
| Suitable Target Perception = U(-1,1) | The value in suitable target can increase or decrease the suitability or leave it unchanged. This enables the offender to sometimes decide a target is not suitable even when they have more wealth. |

[19] Since the jobs data are from 2002, the corresponding year's unemployment rate is employed.
[20] A request to the Seattle Police Department for the average amount of cash taken during street robberies remains unanswered.

Six of the parameters in the model apply to society and include the numbers of civilian and police agents, the unemployment rate, the rate of criminal propensity, the time an offender waits before committing another offense, and the random number seed. The number of civilians and police had to be large enough to ensure that two or more agents would sometimes end up at the same one of the 16,035 nodes but small enough to be computationally feasible. The choice of 1,000 civilian agents and 200 police agents met that balance. In accordance with the criminal careers literature, 20% of civilian agents are assigned a 'readiness' to commit a crime that is positive; making them the only civilians who evaluate each situation for its potential to commit a street robbery. Since even motivated offenders do not offend continuously, a minimum time of one hour is required before an offender can commit another robbery.

Another four characteristics describing time away from home, initial wealth, amount of wealth received each payday, and amount of wealth exchanged during a robbery are set for each agent. Amount of time to spend away from home and initial wealth are assigned using random normal distributions.[21] Amount received each payday and amount of wealth exchanged are fixed. These parameter values are admittedly unrealistic but are chosen to provide a starting point for the research.

Two parameters are important to the decision to commit a robbery because they represent the offender's perception of the characteristics of a situation. The informal guardianship perception value can: 1) magnify the perception of the other agents as capable guardians; 2) reduce it or 3) leave it unchanged. This introduces a stochastic

[21] The choice of distribution (e.g. normal, poisson, etc.) and the mean and standard deviation used to assign values affect the allotment of the characteristics across all the agents.

element into the offender's perception of whether other civilians at the node are capable guardians or not. The suitable target perception value serves the same function for the decision about whether a suitable target exists and enables the offender to decide a target is not suitable even when the target has more wealth.

## *4.5 Agents in the Model*

There are two types of agents which represent people in the model, police and civilians. Police agents have only one role, that of a formal guardian. Lack of a formal guardian in routine activity theory is one of elements that are necessary for a crime to occur. Thus in the model, the presence of a police agent prevents a crime from occurring. To accomplish their mission of crime prevention, police agents randomly move along streets. Police never commit crimes in this model, and they are never targets.

The civilian agents represent the general population of the city. At the start of the simulation, all civilian agents are randomly assigned a starting home location, a wealth level, a criminal propensity indicator, and an allocation of time to spend away from home. Per routine activity theory, a civilian agent can assume one of three possible roles, offender, target, or informal guardian. The particular role a civilian agent assumes is driven by their individual characteristics and the contextual dynamics of the specific interaction. All civilian agents begin each day at home where, by definition, they cannot be involved in street robbery either as a victim or offender. After they spend their allotted time at home, they travel for the rest of the day. Wealth is included in the model as the basis for determining whether the civilian is a suitable target. Every two weeks, all civilian agents receive a static amount of wealth set at five units.

Each of the civilian agents is randomly assigned a percentage of time to stay away from home. While the time each agent spends away from home is unique, the time for society is controlled. This agent characteristic is the basis for the experiments testing societal trends in time spent away from home. Table 3-5 provides an example for the 40% experimental condition. A random normal distribution is used to assign each of the civilian agents a percentage of time to be away from home so that the mean for society is 40%.

Table 3-5: Sample Assignment of Time of Away from Home for the 40% Experimental Condition

| | **Away*** | **At Home** |
|---|---|---|
| Agent 1 | 26% | 74% |
| Agent 2 | 45% | 55% |
| Agent 3 | 60% | 40% |
| Agent 4 | 81% | 19% |
| Agent 5 | 53% | 47% |
| Agent 6 | 30% | 70% |
| Agent 7 | 25% | 75% |
| Agent 8 | 11% | 89% |
| Agent 9 | 41% | 59% |
| Agent 10 | 28% | 69% |
| **Average** | **40%** | **60%** |

*Time 'Away' from home is the difference between total time and time spent at home.

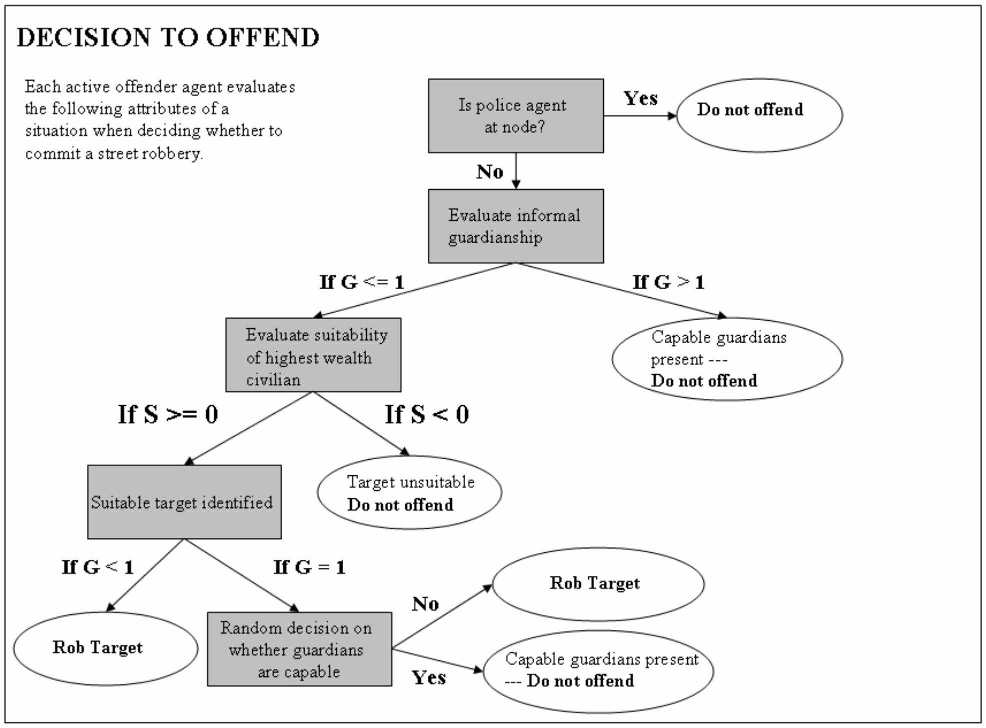
Since routine activity theory recognizes the importance of the frequency with which the elements of a crime converge in generating crime events but does not elaborate on the space-time structure of human activity, this model of crime events characterizes both the distribution and movement patterns of individuals as random. Both the civilian and police agents are distributed randomly across the street intersections in Seattle. Once their time at home is complete, civilians follow a 'random walk' in which the agents move one randomly chosen node each minute of the model (Chaitin, 1990). When an agent (civilian or police) is moving, each adjacent node has an equal chance of being selected and the civilian can backtrack as well as go forward along the network. Civilian agents with criminal propensity travel along the same street network and visit the same locations as other civilians and as police agents. To illustrate the dynamics of the decision to offend, more detail is provided in the next section regarding the behavior of agents with criminal propensity.

## *4.6 Decision to Offend*

At each tick of the model, only those agents with criminal propensity (one at a time) who are traveling consider the following aspects of their situation (Figure 3-2):

1) Is there a police agent at the node?
2) What is the level of informal guardianship at the node?
3) Is there a suitable target at the node?

Figure 3-2: Steps in Decision to Offend


DECISION TO OFFEND
Each active offender agent evaluates the following attributes of a situation when deciding whether to commit a street robbery.
Is police agent at node?
Yes
Do not offend
No
Evaluate informal guardianship
If G <= 1
If G > 1
Evaluate suitability of highest wealth civilian
Capable guardians present --- Do not offend
If S >= 0
If S < 0
Suitable target identified
Target unsuitable Do not offend
If G < 1
If G = 1
Rob Target
Random decision on whether guardians are capable
No
Rob Target
Yes
Capable guardians present --- Do not offend

The level of guardianship and the availability of a suitable target are evaluated via two equations. For computational reasons, guardianship is the first situational element considered by the active agent. If there is a police agent at the same node, the active agent decides not to offend because of too much formal guardianship. However, if there are no police agents and there is at least one other civilian agent at the node, the level of informal guardianship is evaluated further via the formula below (1). First, the total number of civilian agents at the location is evaluated minus the active agent and the potential target. This adjustment reflects the unlikelihood that an offender would act as a potential target's guardian and the inability of the target to be its own guardian. Uncertainty in how offenders perceive the 'capableness' of the other civilians is incorporated into the formula through the addition of a stochastic term $P_G$ that can either increase or decrease the active agent's perception of the level of guardianship in a situation.

$$G = ((N_A - 2) + P_G) \tag{1}$$

If $G < 1$, then there is a lack of capable guardians so condition evaluates to True
If $G = 1$, then make a random decision – condition could evaluate to True or False
If $G > 1$, then capable guardianship is present so condition evaluates to False

Where:
$G$ = Guardianship
$N_A$= number of agents at node
$P_G$ = Perception of capability of guardians who are present (uniform random number between -2 and 2)

In reality, the presence of capable guardians is most likely evaluated along a continuum. On one end of the continuum a police officer is present on the street corner. At the other end, the potential offender is alone with a suitable target. More frequently, situations are somewhere in between.

Finally, the active agent considers whether there are suitable targets at the node. All other civilians who are away from home and at the same node are evaluated using wealth as the primary criteria for identifying a suitable target (2).

$$S = (W_T) - (W_A) + P_S \tag{2}$$

If $S >= 0$, then there is a suitable target so condition evaluates to True
If $S < 0$, then no suitable target present so condition evaluates to False

Where:
$S$ = Perceived suitableness of target
$W_T$ = wealth of target
$W_A$ = wealth of active agent (potential offender)
$P_S$ = Offender perception of target suitability (uniform random value between -1 and 1)

If at least one other agent's wealth exceeds the active agent's wealth, the evaluation of the civilian with the highest wealth continues via the formula above.

The error term $P_S$ represents the influence of other factors on the offender's perception of the relative suitableness of a target and its value can either increase or decrease the perceived suitability of the target. It is worth noting that other agents with criminal propensity who are at the node are included in the active agent's evaluation and can become victims. If $S < 0$, there is not suitable target at the node, and the active agent does not commit robbery.

To recap, for situations in which there is a suitable target, the decision to offend hinges on the level of informal guardianship. If G = True, then there is there is a lack of capable guardians so the decision is to rob the suitable target identified. If G = False, the amount of guardianship is too high, and the decision is not to offend. But if G = 1, the decision could go either way. In these cases, the active agent makes a random decision whether to commit the street robbery. When an agent commits a robbery, one unit of wealth is taken from the victim and transferred to the offender. Once each civilian with criminal intent has evaluated their situation, the model time advances, agents move and the decision structure is repeated.

## *5.0 Analysis*

Both traditional and spatial analysis techniques are used to examine the results of the model runs. Descriptive statistics such as mean, median and standard deviation are used to characterize the results of each of the experiments. As is customary practice, an ANOVA is applied to determine if there is a significant difference among the RobRates for the five experimental conditions (Axelrod, Forthcoming). The number of robbery victimizations for each civilian agent in the model is the response variable. A sample size of five thousand observations across five experimental conditions provides a very powerful design. Thus increasing the sample size to ten thousand or twenty thousand should not change any findings of significant differences among the groups. Finally, the resulting spatial patterns of robbery events are examined.

At the agent level, descriptive statistics are generated to test the relationships among time spent away from home (AwayTime), total number of victimizations (TotVict), and total number of robberies committed (TotOff). These statistics are examined for the total population and then just for agents with and without criminal propensity.

Two approaches to describing the spatial distribution of street robberies are taken. A visual comparison is made of the resulting crime patterns using a kernel density. Kernel density surfaces offer a means of evaluating the existence of global trends in the distribution of street robberies and for comparing the relative density of robberies across experimental conditions. To create a kernel density, a temporary grid is laid over the entire study area and a density value for each cell in the grid is computed using a circular 'neighborhood' (Bailey & Gatrell, 1995; Mitchell, 1999; Williamson et al., 2001).[22]

In addition to the kernel density, formal tests of the spatial distribution of crime events are employed using Ripley's *K* function. Ripley's *K* is applied to compare the clustering of robberies and visits to places at different scales. Typically, the *K* function for complete spatial randomness (CSR) is helpful in identifying whether the observed pattern is significantly different than what would be expected from a

[22] The term kernel refers to size of the 'neighborhood' (also called bandwidth) that is taken into account when computing the density. The total number of street robberies within the bandwidth are summed and divided by the area under the circle. The resulting value is assigned to the current cell.

random distribution (Bailey & Gatrell, 1995; Levine, 2005). A known weakness of comparing the observed distributions to CSR is that most human-generated patterns are non-random (e.g., population, housing, etc.) (Levine, 2005).

In this research, CSR cannot be used to evaluate the clustering in street robbery events because the locations at which data are collected are constrained to a fixed set of locations representing the intersections in Seattle.[23] Since the CSR algorithm randomly places points anywhere within the study area boundary, it would be inappropriate to compare the clustering in robberies and number of visits, which are constrained to the street nodes, to a randomly generated CSR. However, a *K* function can be generated from the pattern of street nodes thus revealing the extent of the clustering intrinsic to the street network. Comparing the *K* function for street intersections to CSR answers the question of whether the intersections are more clustered than would be expected under CSR. Taking this one step further, the *K* function for street robberies can be compared to the one for intersections to find out if robberies are more clustered than the street intersections.

Another aspect of the same discussion involves the role of the street nodes in structuring the initial distribution of police and civilians since they too can only be allocated to a street node (and not to any location within Seattle). In this way, the structure of the street network conditions both the original distribution of agents and their movement. Since the agents are randomly assigned to nodes and they move randomly during the simulation, the distribution of robberies should be similar to that of the network nodes if space alone determines where street robberies occur. To

[23] Thanks to Ned Levine for pointing out this issue.

check this, the *K* function for nodes is compared to the *K* functions for both robberies and visits.

## *6.0 Findings*

The creation of the street robbery model enables the exploration of routine activity theory's propositions via simulation. Two research questions are addressed in the analysis. The first asks whether the shift in routine activities away from home increases the incidence of street robbery. The second examines the spatial pattern of street robberies as members of society spend more time away from home. This section describes the behavior of the model and summarizes the findings of the tests. The robustness of the findings is then evaluated by running the model using five different random number seeds and systematically varying key parameter values for each seed.

### 6.1 General Description of Model Outcomes

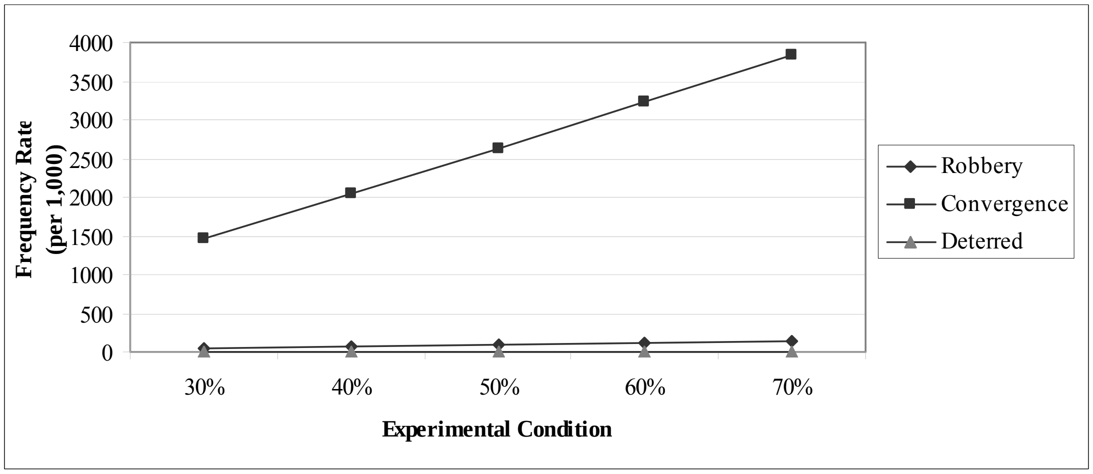
Data describing nine attributes of places and society are collected to characterize the results from the model runs across the five experimental conditions. Societal-level changes in the number of street robberies and convergences of agents in space-time (i.e. opportunities for street robbery) are in line with what routine activity theory would predict; both values increase with time spent away from home (Table 3-6). The number of times the presence of a police agent prevents a robbery from taking place also increases as the societal time spent away from home increases. In all likelihood, this increase in deterrence is directly related to the existence of more situations in which a crime might occur (i.e. a motivated offender and suitable target are at the same place-time).

Table 3-6: Societal-level Model Outcomes

| | Experimental Condition | | | | |
|---|---|---|---|---|---|
| | Cond 30 | Cond 40 | Cond 50 | Cond 60 | Cond 70 |
| **Societal-Level** | | | | | |
| Total Robberies | 54,637 | 76,032 | 95,219 | 118,085 | 139,007 |
| Total Intersections | 1,454,341 | 2,050,761 | 2,631,149 | 3,238,760 | 3,835,299 |
| Total Robberies Deterred by Police | 1,532 | 2,148 | 2,693 | 3,430 | 4,040 |
| Model time at home (minutes) | 1003.09 | 859.79 | 716.51 | 573.21 | 429.91 |
| Model time spent away (minutes) | 436.91 | 580.21 | 723.49 | 866.79 | 1010.09 |

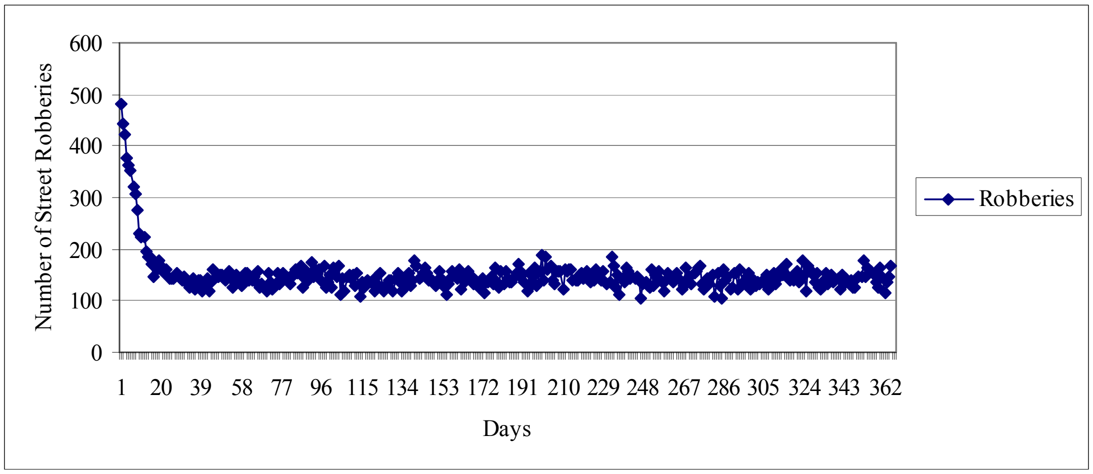
More in-depth examination reveals the number of intersections, street robberies, and robberies prevented by police across experimental conditions are related (Figure 3-3). As time away from home increases, the number of convergences displays the highest rate of increase, followed by the number of robberies, and the number of robberies deterred by police presence.

Figure 3-3: Comparison of Robbery Incidents, Convergences and Robberies Deterred



Examining the amount of time before the model reaches equilibrium provides insights into model dynamics. Figure 3-4 illustrates the change in the number of robberies per day over the entire model year for the 30% condition. The first day has the highest number of robberies (N=444). The number drops rapidly each day until about day 24 when equilibrium is achieved at around 150 incidents per day. During equilibrium the number of robberies per day fluctuates between 100 and 200 for the rest of the year. So the major changes in the model are occurring before the end of the first month, even though the wealth of individual agents continues to change via robbery events and paydays throughout the model year.

Figure 3-4: Days to Equilibrium for Street Robbery Simple Model (condition: 30%, seed = 100)



One potential explanation for this pattern lies in the way wealth is distributed. The same initial wealth distribution is applied regardless of criminal propensity and each civilian agent receives the same amount each payday. However, agents with criminal propensity can also gain wealth from committing robberies. Over time, this wealth advantage translates into higher levels of wealth for offenders as compared to non-offenders. Evidence of this growing wealth imbalance begins around day 24.

When the agents with criminal propensity tend to have more wealth, fewer civilians qualify as suitable targets. Robberies become restricted to situations in which there are two or more agents with criminal propensity because only in these situations is there likely to be an agent with more wealth than the active offender. This reduces the number of robberies experienced by society, although the convergences remain high. The model process just described, while informative in terms of understanding model dynamics, does not impact the comparison among model conditions because the behavior of the process is consistent across experimental conditions. The more time spent away from home the fewer days it takes to reach equilibrium and the higher the equilibrium number of robberies.

## 6.2 Testing Routine Activity Theory

A One-Way ANOVA is applied to test the hypothesis that robberies will increase as time spent away from home increases. By comparing the mean number of robberies across the five experimental conditions, it is possible to determine if robbery increases as the time spent away from home increases. The results of the ANOVA indicate there are there are significant differences on the rates of street robbery across the experimental conditions (Table 3-7).[24] However, the test does not provide information on which of the conditions were significantly different.

[24] Because of the positive skew to the distribution of robberies, additional tests regarding the equality of means were conducted. Both the Brown-Forsythe and the Welch tests for equality of the means are significant at .000. These tests are preferable to the F test when the equality of variances assumption is violated as it is here ("SPSS for Windows," 2002).

Table 3-7: Change in Street Robbery Events across Experimental Conditions

| | Condition | | | | |
|---|---|---|---|---|---|
| | **30%** | **40%** | **50%** | **60%** | **70%** |
| Target time to spend away from home in minutes (hours) per day | 432 (7.2) | 576 (9.6) | 720 (12) | 864 (14.4) | 1008 (16.8) |
| Actual time spent away from home | 436.9 | 580.2 | 723.5 | 866.8 | 1010.1 |
| Number of civilian agents (N) | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 |
| Mean robberies (Standard Deviation)*** | 54.64 (101.99) | 76.03 (144.15) | 95.22 (182.35) | 118.09 (228.14) | 139.01 (270.06) |

*** Difference among one or more of the groups is significant at $P < .000$.

Results from post hoc tests employed to identify which experimental conditions are significantly different from one another are inconsistent (Table 3-8).[25] Beginning with the 30% condition, there is significantly less crime in societies in which individuals spend 30% of their time away from home when compared to each of the other conditions. Other significant differences are between the 40% condition and both the 60% and 70% conditions; as well as between the 50% and the 70% conditions.

[25] The Levene statistic is significant indicating the variances are significantly different among the groups. However, ANOVA is robust in the face of this violation when the group sizes are equal which they are in this research (Newton & Rudestam, 1999; Shannon & Davenport, 2001). A Tamhane's T2 post hoc test is used because it does not assume equal variances.

Table 3-8: Post Hoc Tests of Mean Differences (seed = 100)

| *(I) Randomization Condition* | *(J) Randomization Condition* | *Mean difference (I - J)* | *Standard error* | *Significance* |
|---|---|---|---|---|
| 30% Time away | 40% Time away [a] | -21.39 | 5.584 | .001 |
| | 50% Time away [a] | -40.58 | 6.607 | .000 |
| | 60% Time away [a] | -63.45 | 7.903 | .000 |
| | 70% Time away [a] | -84.37 | 9.129 | .000 |
| | | | | |
| 40% Time away | 50% Time away | -19.19 | 7.351 | .088 |
| | 60% Time away [a] | -42.05 | 8.534 | .000 |
| | 70% Time away [a] | -62.98 | 9.681 | .000 |
| | | | | |
| 50% Time away | 60% Time away | -22.87 | 9.236 | .126 |
| | 70% Time away [a] | -43.79 | 10.305 | .000 |
| | | | | |
| 60% Time away | 70% Time away | -20.92 | 11.180 | .470 |

[a] Significant differences were found between experimental conditions *I* and *J*.

### 6.3 Spatial Distribution of Street Robberies across Places

The spatial distribution of street robberies is addressed via descriptive statistics, examination of the outcome pattern, and quantitative description of the concentration of robbery events. The spatial distribution of agent movement and robberies across intersections reveals both increased concentration and a slight increase in spread, as time spent away from home increases (Table 3-9). Looking first at the summary statistics for places in Seattle, the mean visits per intersection increases at the same rate across all experimental conditions as does the mean robberies per intersection. However, both the percentage of intersections with only one robbery and those with more then one robbery have their largest increase between the 30% and 40% conditions.

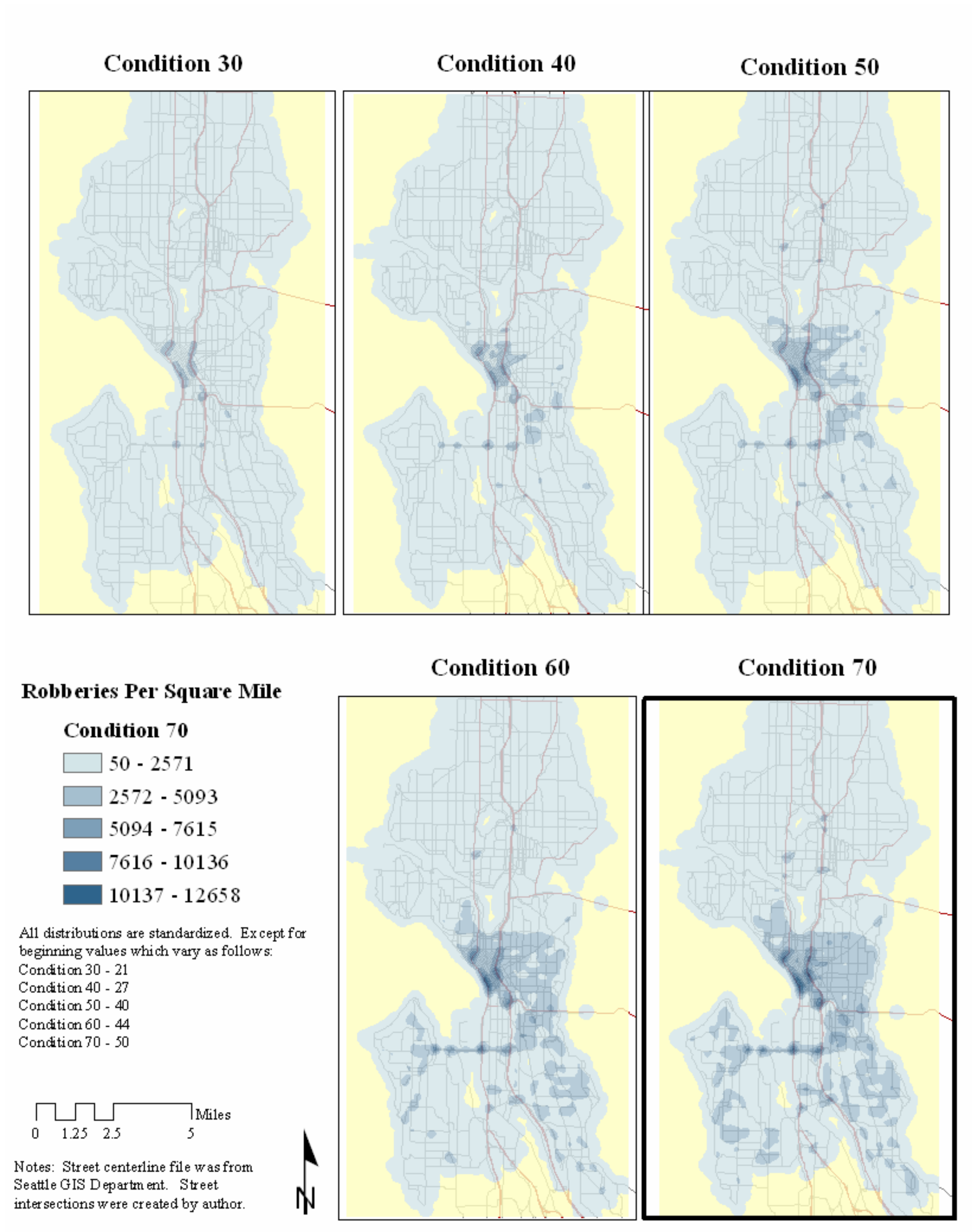
Table 3-9: Place-level Model Outcomes

| | Experimental Condition | | | | |
|---|---|---|---|---|---|
| | **Cond30** | **Cond40** | **Cond50** | **Cond60** | **Cond70** |
| **Place-level** | | | | | |
| Mean visits per street node | 9,733 | 12,904 | 16,088 | 19,253 | 22,423 |
| Mean robberies per street node | 3.41 | 4.74 | 5.94 | 7.36 | 8.67 |
| Percent of street nodes with a robbery | 83% (N=13,376) | 87% (N=13,925) | 89% (N=14,309) | 91% (N=14,531) | 92% (N=14,683) |
| Percent of street nodes with more then one robbery | 70% (N=11,157) | 76% (N=12,175) | 81% (N=12,995) | 83% (N=13,303) | 85% (N=13,572) |

The spatial pattern of robberies is examined across all five conditions using kernel density (Map 3-1).[26] A visual inspection of the map series indicates support for the second hypothesis. At 30% time spent away from home, a few areas of concentration appear. As civilians spend more time away from home, the densities of those original concentrations increase while new areas of higher density appear. This pattern reflects both the increased frequency of the convergence of the elements necessary for a crime to occur and the larger travel areas of agents as they spend more time away from home.

[26] A bandwidth of 1,320 feet (one quarter mile) and a cell size of 100 feet are the basis for all kernel density surfaces. The quarter mile distance is often employed to represent the potential walking area for individuals in urban areas and by extension their potential area of interaction (Calthrope, 1993; Duaney & Plater-Zyberk, 1993; Nelessen, 1994). The surfaces are generated in ArcGIS version 9.1 and the output is in robberies per square mile (Mitchell, 1999).

Map 3-1: Kernel Densities for Modeled Street Robbery Events

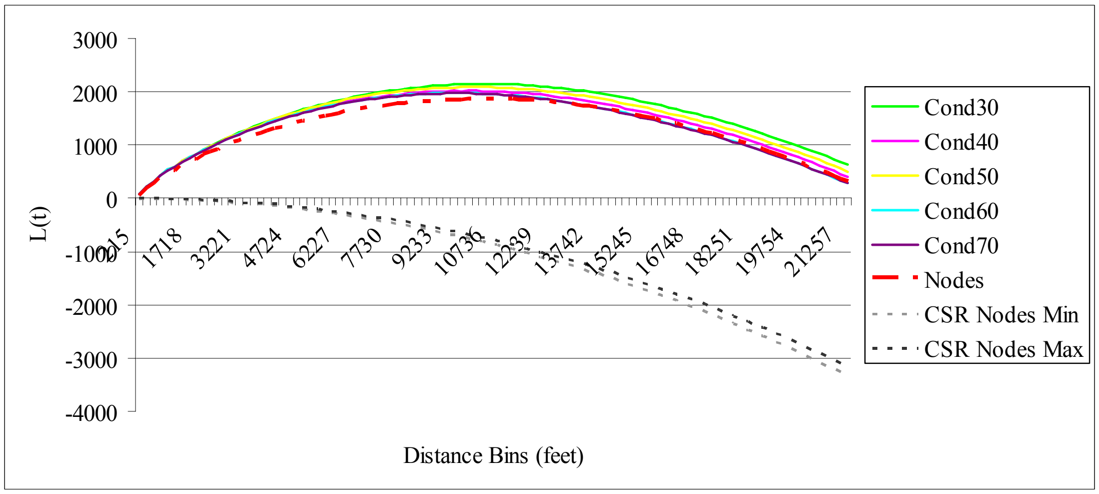


All distributions are standardized. Except for beginning values which vary as follows:
Condition 30 - 21
Condition 40 - 27
Condition 50 - 40
Condition 60 - 44
Condition 70 - 50

Notes: Street centerline file was from Seattle GIS Department. Street intersections were created by author.

Results of the Ripley's K function indicate that there is a high degree of concentration in street robbery locations across all five conditions.[27] Figure 3-5 compares the concentration of street robberies generated from each of the five experimental conditions to the concentration of the street network's nodes and to a reference distribution describing the amount of concentration that would be expected under CSR.[28] Concentration in street nodes increases until approximately one and one-half miles when it levels off for about a half mile and then begins to decline. The graph reveals that street nodes are significantly more concentrated than would be expected under CSR.

Figure 3-5: Ripley's *K* for Robbery across Experimental Conditions



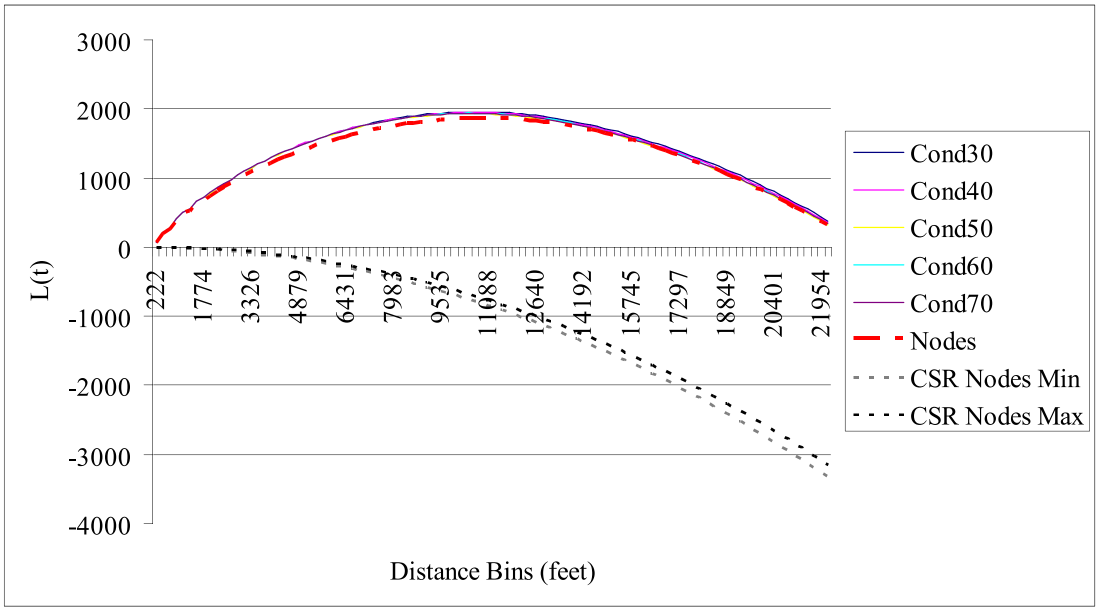
The street robbery distribution lines for each experimental condition follow a similar pattern to the line for all street nodes. All six lines are identical until about 800 feet when the robberies become more clustered than the street nodes. This

[27] The reported Ripley's *K* functions are generated using CrimeStat III. No edge correction is applied since approximately three quarters of the perimeter of Seattle is bounded by water.
[28] The CSR *K* function distribution is generated by using a uniform random number generator to create 100 distributions with the same N as the observed distribution, in this case N=16,035 (Levine, 2005). A significance level of $p < .05$ is used. The random distribution generated under CSR is truly random in that any location can be selected, not just an intersection.

pattern continues until about 2.25 miles when the robberies in the 60% and 70% conditions converge with the line for street nodes. The lines for the 30, 40 and 50% conditions remain more concentrated than the street nodes at all distances and the difference between concentrations is consistent. Robberies are most concentrated when society spends 30% of time away from home, and the concentration decreases as time spent away from home increases.

Data characterizing the total number of visits experienced by each node offer a means of quantifying agent travel patterns. Comparing the distribution of all agent movement with the distribution of robberies provides a test of whether the two distributions are different. Figure 3-6 suggests that the pattern of visits across nodes is very similar to that of street robberies. However, the lines describing visits are closer together indicating there is even less difference in the concentration of the distributions for visits than there is for street robberies. These results suggest that street robbery incidents tend to occur where many agents routinely converge but that robberies have other factors contributing to their greater concentration that are not accounted for by the structure of the street network.

Figure 3-6: Ripley's *K* for Visits across Experimental Conditions



Overall, the results of the K function suggest support for the important role that the street network plays in the concentration of street robbery. Specifically, they illustrate that street nodes are significantly more concentrated than would be expected under CSR and that their intrinsic clustering is responsible for the majority of clustering in both agent travel and street robberies. This finding underscores the importance of considering the street network in any evaluation of the concentration of travel (i.e. visits) and street robberies that are produced by the model. That being said, the results also suggest that there are situational factors at work in generating the observed robbery patterns. Even though the agents (both civilians and police) are randomly distributed across street intersections at the beginning of the simulation and then move randomly from node to node over the whole model year, there is additional clustering in the distributions of street robbery events that cannot be accounted for by the pattern of street nodes.

## 6.4 Some Comments on the Robustness of the Model

Sensitivity testing is essential to quantifying the robustness of the model results and is conducted by varying the initial parameters and the random number seed (Manson, 2001). The values of five of the parameters (i.e. number of police, time to wait before able to re-offend, initial wealth distribution, perception of target suitability random term and the perception of guardianship random term) are increased; the model runs repeated for all five experimental conditions; and a one-way ANOVA applied to analyze the results. While the absolute number of street robberies increased or decreased depending on the parameter being varied, in all cases the original significant differences between the groups remained, demonstrating the robustness of model results to changes in initial parameters (Table 3-10). Finally, the entire sensitivity testing process of varying the five parameter values is repeated four more times using different random number seeds to test the effect of changing the random number seed on the outcomes of the model. An analysis of the output demonstrates that model results are robust to changes in the random number seed.[29]

[29] The results of the sensitivity tests with random number seeds of 200, 300, 400 and 500 are available upon request.

Table 3-10: Parameter Testing Results

| | Condition | | | | |
|---|---|---|---|---|---|
| | 30% | 40% | 50% | 60% | 70% |
| Target time to spend away from home (hours) | 432 (7.2) | 576 (9.6) | 720 (12) | 864 (14.4) | 1008 (16.8) |
| Actual time spent away from home | 436.9 | 580.2 | 723.5 | 866.8 | 1010.1 |
| | | | | | |
| Base Model (robbery rate per 1000 agents)*** | 54.637 | 76.032 | 95.219 | 118.085 | 139.007 |
| | | | | | |
| Parameter Verification | | | | | |
| Increase number of police to 1000*** | 54.148 | 72.502 | 91.474 | 110.752 | 131.611 |
| Increase time to wait before re-offending to one day*** | 31.682 | 37.358 | 41.403 | 44.551 | 47.134 |
| Increase societal wealth distribution (mean = 100 and sd=50) *** | 58.937 | 79.097 | 100.195 | 121.916 | 143.241 |
| Increase impact of random term representing perception of target suitability U(-10,10) *** | 42.093 | 56.688 | 73.931 | 87.699 | 103.353 |
| Increase impact of random term representing perception of guardianship U(-4,4) *** | 51.427 | 72.520 | 89.590 | 108.469 | 128.618 |

*** Difference among one or more of the groups is significant at P < .000.

## *7.0 Discussion*

This paper presents a new approach to formalizing and testing criminological theory that relies on simulation. To demonstrate the methodology, a simulation model of street robbery is developed based on the core propositions of routine activity theory. The model is then used to conduct a series of experiments to test whether the outcomes match what the theory predicts.

Previous attempts to test routine activity theory, although generally supportive, have produced mixed results. None of the tests were able to sufficiently address the spatio-temporal structure of routine activities, satisfactorily deal with measurement issues, or effectively capture the dynamic nature of interactions at the micro-level. This research addresses all three of those issues by relying on simulated individuals that interact on the streets of Seattle, Washington.

Routine activity theory's basic premise, crime will increase as individuals spend more time away from home, is strongly supported by model results and the finding is robust even when the initial parameter values are systematically varied. Although the absolute number of robberies fluctuates as parameters are changed, the relative relationship between increasing time spent away from home and the rate of street robberies remains significant.

Previous research has recognized the role of the built environment in general to structuring movement (Capone & Nichols, 1976; O'Sullivan, 2004a) and to concentrating population-related variables (Bailey & Gatrell, 1995; Levine, 2005). This research finds the street intersections are significantly more clustered than would be expected by chance. Taking this clustering into account, the pattern for street robberies exhibits additional clustering beyond what is explained by the street network but only at certain distances. At these distances, the clustering is instead related to the specific situation in which the crime occurs.

Although this initial implementation of the method is simple, it accomplishes several essential functions. First, it makes the process of theory testing transparent by formalizing model specifications. Subsequent researchers have a concrete record of how theoretical constructs are operationalized in the model. Second, the model provides a base upon which to build more complex explorations of street robbery. Third, the method replaces artificial landscapes prevalent in agent-based models with the street network of Seattle. In doing so, the research takes an important step toward more realistically 'situating' simulation and measuring the street network's influence on spatial patterns of street robbery. Fourth, the use of a series of controlled experiments to test the model illustrates the potential for this type of research to refine theory by systematically varying one aspect while holding all others constant.

New questions could be explored by building additional analytical capability into the base model. The incorporation of activity spaces for civilians represents an important and necessary enhancement to the initial model. Rather than traveling randomly, individuals could be assigned home, job, and other locations among which they could travel (Groff, Forthcoming-b). The spatial distribution of homes, jobs, recreation and services in Seattle could serve as the basis for the distribution of agents' activity nodes in the model. In this way, the activity locations of agents in the model would reflect the activity spaces of the civilians of Seattle.

The behaviors and awareness levels of agents could be expanded and made more nuanced. Enhancing the behavior of existing police agents would enable tests of the effectiveness of different patrol strategies (e.g. hot spot policing) in reducing or displacing crime. For example, a researcher could compare the results of the previous simulation in which police patrol randomly to a hot spots policing strategy in which police are assigned areas where street robbery is highest. In addition, a wider range of place characteristics and neighborhood-level perceptions of areas could be incorporated into the dynamic decisions of individuals. This would enable more richly textured micro-level situations in which agents interact as well as incorporate important micro and macro-level elements that impact how a situation is perceived.

At the individual level, the influence of guardianship on the decision to offend could be studied intensively. Specifically, the role of place managers and intimate handlers as guardians could be tested in a computational laboratory (J. E. Eck, 1995a; Felson, 2001;2002). For example, a researcher could change the weightings of different types of agents (e.g. police, known agents, place managers etc.) to determine the effect on the decision to offend while holding everything else constant.

## *8.0 Conclusion*

Despite the potential value of simulation as a research platform, serious questions remain about evaluating models (Manson, 2001; O'Sullivan, 2004a). Focusing on simulation models as tools to aid in explanation and understanding rather than prediction avoids many of the thorniest questions of model validation. In this role, simulation models become aids to the refinement of theory prior to empirical testing and are especially useful in identifying 'gaps' in theory (O'Sullivan, 2004a). In addition, simulation models have the potential to reduce research costs by saving empirical tests for the strongest theories. After all, simulated theory testing takes place in an artificial world and thus is not capable of conferring empirical validity to a theory (Paternoster, 2001).

The research methodology employed here provides a unique framework for promoting more comprehensive and rigorous tests of theories about human behavior at both the micro- and macro-levels of analysis. Because the method requires the formalization of theoretical concepts, it has the potential to generate a common language with which to describe those concepts, and to stimulate the construction of well-defined models that can be discussed and tested further. The findings from the example model of street robbery specified and tested here demonstrate clear support for the plausibility of the basic premise of routine activity theory and in doing so provide the foundation for the development of additional, more richly specified models of criminal and spatial behavior. Advanced models are likely to produce concrete, public policy relevant findings addressing both the situational elements of crime and the structure of routine activities in general. The potential for crime reduction from these findings is high because the situational aspects of the crime event can be altered far more quickly and easily than ones involving the root causes of criminal motivation (Akers, 2000; Cullen & Agnew, 1999; Felson, 1987; Vold et al., 2002).

# Chapter 4: The Spatio-Temporal Aspects of Routine Activities and Crime

## *1.0 Introduction*

Researchers within geography and closely related disciplines have long recognized the importance of considering space and time when examining human behavior (Chorley & Haggett, 1967; Engel-Frisch, 1943; Hägerstrand, 1970;1973; Harvey, 1969; Hawley, 1950; Horton & Reynolds, 1971; H. J. Miller, 1991; R. J. Sampson, 1993). In particular, sparked by theoretical developments during the 1970s and 1980s, many criminologists have begun to study how places influence when and where victims and offenders converge (D. L. Weisburd, 2002). Proponents of this view focus on the study of crime events rather than criminal motivation and rely on a set of 'opportunity theories' of crime to explain why crimes occur in one place and not another.

As the importance of place and time in criminological theory has gained recognition, so has the utility of applying a more process-oriented perspective to the study of crime (R. J. Sampson, 1993; D. L. Weisburd, Lum, & Yang, 2004). This approach recognizes that "social behavior occurs in particular times and places with particular social actors" (R. J. Sampson, 1993 429). While the theoretical framework exists for such research, the collection of individual-level data to characterize human interactions in general and crime events in particular remains an on-going barrier to the empirical application of this perspective and one that is unlikely to change due to privacy concerns (O'Sullivan, 2004b).

In response to these challenges, some researchers have begun to consider simulation modeling as an alternative approach (P. L. Brantingham & Brantingham, 2004; J. Eck, 2005; J. E. Eck & Liu, 2004; Gilbert & Terna, 1999; Gilbert & Troitzsch, 1999; Gimblett, 2002; Liu et al., 2005; Macy & Willer, 2002; Moss & Edmonds, 2005). A subset of these researchers are interested in crime and recognize the value of simulation modeling for: 1) understanding crime in its situational context; and 2) capturing the dynamic interactions taking place at the micro level and examining their relationship to macro level patterns (P. L. Brantingham & Brantingham, 2003;2004; P. L. Brantingham & Groff, 2004; J. E. Eck & Liu, 2004; Gunderson & Brown, 2003; Wang et al., 2004). In particular, the Brantinghams (2004) have clearly illustrated the important role of agent-based models in formalizing the context in which a crime event occurs.

Many researchers use simulation modeling for prediction and forecasting (Maguire et al., 2005; E. J. Miller, Hunt, Abraham, & Salvini, 2004; E. J. Miller, Roorda, & Carrasoc, 2005). Others emphasize the potential of simulation for elaborating theory (Albrecht, 2005; J. Eck, 2005; Macy & Willer, 2002) and for conducting systematic experiments in virtual laboratories (Dibble, 2001; Epstein & Axtell, 1996; Gilbert & Terna, 1999; Macy & Willer, 2002).

One recent study offers an example of combining theoretical exploration with controlled experiments to study the crime of street robbery. This was accomplished by implementing the assumptions of a theory, in this case routine activity theory (RAT) (Cohen & Felson, 1979), in a simulation model and then testing them via controlled experiments to discover whether the theoretically-predicted outcomes match the model outcomes (Groff, Forthcoming-a). The model building process emphasized simplicity, focusing on the elements that were directly addressed by the theory (Macy & Willer, 2002) and relied on 'situating' simulation by combining agent-based modeling (ABM) with geographic information systems (GIS) to include space and time. The study found support for RAT's core proposition that shifts in routine activities away from home increase the incidence of street robbery. In addition, a spatial analysis demonstrated that the observed clustering in street robbery events is beyond the degree that would be expected based on the configuration of the streets. The approach taken in the study represents a middle ground for theory 'elaboration' between the verbal formulation of the theory and the testing of theory with empirical data (J. Eck, 2005); some characterize it as a way of 'experimenting on theories' (Dowling, 1999).

RAT's recognition of the importance of space and time in determining the convergence of the elements and the simplicity of the theory make it an ideal candidate for underpinning a further exploration of temporal and spatial impacts on crime patterns. This research extends the earlier study by adding defined temporal and spatio-temporal schedules to the base model of street robbery. Systematically adding complexity to the original model (Groff, Forthcoming-a) makes it possible to isolate the effect of first time and then space on the amount and spatial distribution of street robbery. This is accomplished via the use of a new method for implementing activity spaces in agent-based models for 'situating simulation' to better reflect the influence of the built environment and urban structure on agent behavior (Groff, Forthcoming-b).

The remainder of this paper begins with a brief description of the theoretical basis for both the original model and the space-time extensions. Next, the research design is described including: the application of agent-based modeling to this topic; the input data that are used to characterize agent activity spaces; and the experiments conducted. The implementation model is explained to provide a basis for interpreting the model results. The analyses of model results reveal that adding temporal and spatio-temporal schedules to agent activity spaces significantly alters the incidence and spatial distribution of street robberies. Support for routine activity theory's premise that crime will increase as people spend more time away from home is found in the Temporal version only.

## *2.0 Theoretical Background*

Since this research extends an existing model, the next section only briefly describes the theoretical basis for that model (Groff, Forthcoming-a). The bulk of the current section provides the theoretical background for the spatial and temporal components of human activity and their implications for the frequency and timing of the convergence of individuals.

### 2.1 Criminological Foundations for the Original Street Robbery Model

The original model of street robbery is primarily based on routine activity theory (Cohen & Felson, 1979) but relies on rational choice theory for the specifics of offender decision-making (Clarke & Cornish, 1985;2001). This is necessary because routine activity theory pays little attention to the source of the offender's motivation and merely assumes a supply of motivated offenders. None of the later extensions to RAT are considered.

The central premise of Cohen and Felson's (1979) routine activity theory (RAT) is that increases in crime are the result of a shift of routine activities away from home. As originally conceptualized, RAT identifies the convergence of *motivated offender*, *suitable target*, and the *lack of* a *capable guardian* at a particular place and time as the core elements necessary for a crime to occur. A fourth element, *routine activities,* influences when and where victims and offenders converge. Routine activities are the key dynamic element in determining aggregate crime rates because this element affects the convergence of the three other elements necessary for a crime, motivated offender, suitable target and guardianship. In sum, crimes occur when the normal, everyday activities of offenders and victims converge in space and time with no guardian present. Changes in routine activities directly impact the frequency of convergence among these elements which in turn, increase or decreases overall crime rates.

### 2.2. Background on the Spatio-temporal Nature of Human Activity

One of the core concepts in RAT involves the necessity of the convergence of victims and offenders in space and time in order for a crime to occur. The specific 'where' and 'when' of convergence stems from the routine behavior patterns of each actor involved. Thus representing the spatio-temporal aspects of human behavior that facilitate convergence is a critical element in modeling street robbery events since it is the interactions between humans and their environment that serve as the source of explanation of observed spatial patterns (Aitken et al., 1989; R. G. Golledge & Timmermans, 1990; Walmsley & Lewis, 1993).

Scholars have long recognized that capturing only a single dimension leaves more questions than are answered (Hägerstrand, 1970;1975; Pred, 1967; Thrift & Bennett, 1978; Thrift & Pred, 1981) and that examining space-time together yields a different and more complete representation of a situation than studying temporal variation or patterns in space individually (Pred, 1996). However, RAT does not address any specifics of routine activities and the original model (Groff, Forthcoming-a) implements only a simple notion of activity space in which agents were either at home or away from home. A more complex representation of activity spaces is required to capture the space-time aspects of human behavior.

Although modeling activity spaces is not the main focus this research, their creation is essential to representing the *routine activities* component of routine activity theory. Fortunately, a substantial literature exists that is dedicated to the examination of issues surrounding time-space geographies. The time-geographic approach developed by the Lund School in Sweden is highlighted because it focuses on developing a set of probable behaviors not on trying to predict activity spaces based on empirical data and emphasizes increased understanding as a goal (Hägerstrand, 1975). This is accomplished by explicitly representing how the opportunities and constraints within which individuals operate limit the temporal and/or spatial extent of activities. Related approaches to studying space-time patterns relevant to this research are touched upon. Finally, attempts at empirical implementation of time-geographic principles that incorporate the use of GIS are discussed. Together these investigations provide the foundation for the modeling approach taken in this research.

## *Time-Geography as a Framework for Modeling Routine Activity Spaces*

Time-geography, as developed by Hägerstrand (1970; 1975), is a conceptual framework that takes into account the spatio-temporal aspects of human behavior as situated within larger social processes (Thrift & Pred, 1981). Time and space are components of every action and interaction; one cannot be considered without the other. Individuals travel to various locations along *paths*. They operate with a known *domain* and points at which individuals stop their spatial movement (e.g. work, school, recreation) for a time are referred to as *stations*. A large part of human interaction occurs at stations. None of these elements are static, for example, domains and bundles can change as people change jobs or as their circumstances change (Hägerstrand, 1970).

Several aspects of time-geography are important to the current investigation. Individual's travel patterns are influenced by constraints (temporal, economic and spatial) on their ability to take advantage of opportunities for housing, employment, recreation etc. Focusing on individual opportunities and constraints as they play out within a particular context is essential to understanding why events, in this case street robberies, occur sometimes and not others (Pred, 1996). Hägerstrand identifies three main types of constraints that shape both the destinations and the route taken by individuals as they go about their daily activities; *capability*, *coupling* and *authority* (Hägerstrand, 1970). *Capability constraints* relate to physical or resource-related limits to activity. A physical constraint may consist of the time needed to sleep while resource-related constraints are often related to distance. For example, a person who can afford a car has greater mobility than someone who walks. Especially important to this research is how capability constraints interact with fixed locations to limit travel (e.g. individuals can travel no farther away from home or work than the amount of time needed to return). The actual distance depends on several factors including mode of transportation, street network, and speed limits; all of which factor into the amount of distance that can be covered in the time allotted. Thus, models of routine activities must include both individual behavior and urban form if we are to better understand how the two are related and how they together produce human travel behavior (Hägerstrand, 1975).

While the *coupling constraints* and *authority constraint*s are not incorporated into the current research, they represent important information to inform future efforts. *Coupling constraints* involve activities that must be undertaken with others and thus they require both spatial and temporal overlap among specific sets of individuals (e.g. coworkers, customers, and friends). Individuals are also constrained by the time-space aspects of *authority constraints* that regulate access to particular areas at specific times.

As the preceding paragraphs demonstrate, both time-geography and routine activity theory address individual-level behavior in the context of the macro-level environment and thus the two lines of research are complimentary.

### *Implementations of Time-geographic Models*

While time-geography offers good conceptual grounding for understanding human behavior, its implementation to conduct empirical research has faced a variety of challenges. Early efforts, such as that of Lenntorp (1978), were hampered by complications in translating conceptual models into physical data models (Huisman, Forer, & Albrecht, 1997 as summarized from Forer 1993; H. J. Miller, 1991); the lack of data describing the space-time activities of individuals (Huisman & Forer, 1998; H. J. Miller, 2001); and difficulty in representing what data were available in a GIS, especially the temporal element (H. J. Miller, 2001; Peuquet, 1994).

Miller (1991) took a major step toward computer modeling of human behavior when he defined the data inputs necessary for operationalizing time-geographic activity spaces within a GIS. He terms the set of available places and routes a person can visit within a time period a 'potential path area' (PPA). A PPA is limited by the activities that a person needs to accomplish, where those activities are located, and the street network (H. J. Miller, 2001). Most importantly for the current research, he identifies the following elements necessary for defining a PPA: 1) locations that function as the origin/end of trips; 2) locations of activities; and 3) the arcs and nodes describing the travel environment (H. J. Miller, 1991).

In his original work Miller (1991) recognized the potential of GIS for implementing time-geographic principles but noted that the main hurdle has been the incorporation of time into GIS (Peuquet, 1994;2002). More recent work by Miller (2005) argues that the time-geographic conceptual framework cannot provide the specificity necessary to develop physical data models. In response, he offers specifications for a measurement theory for time-geography (H. J. Miller, 2005). Briefly, they involve capturing the location (as an x, y coordinate pair) and the time (as a discrete time slice, t) for every individual in the model. Miller's (2005) methodology takes into account velocity of travel, as reflected by mode and speed limits, as a constraint on the physical distance that can be covered within a specified time. These constraints structure the timing and thus the 'spacing' of discretionary activity, an important element in criminal behavior (Ratcliffe, in press).

### *Other perspectives on human behavior*

Several other perspectives are drawn from to inform agent movement and routine activities in the model. In general, these perspectives agree that each person has an *activity space*; a geographic area within which they conduct their daily activities (Horton & Reynolds, 1971). This area is equivalent to Hägerstrand's (1970; 1975) *domain* and encompasses both the locations that are visited and the paths taken among those locations. Different researchers have their own terms for these locations and paths. Locations that are visited are called *nodes* (Paul Brantingham & Brantingham, 1981b; Patricia Brantingham & Brantingham, 1993; Lynch, 1960; H. J. Miller, 1991) or *anchorpoints* (R. Golledge & Stimson, 1997; G. Rengert, 1988). These are the places where the majority of human interaction occurs. The particular routes taken among the locations are termed *paths* (Lynch, 1960; H. J. Miller, 1991). These paths often represent the shortest route between two places (Felson, 1987).

Regardless of the terminology, these perspectives share the following characteristics. Individuals' travel patterns are influenced by temporal, economic and spatial constraints that limit their choices related to housing, employment, recreation etc. Home tends to be the dominant node and travel is concentrated along certain routinely frequented paths. Frequently traveled paths are hypothesized to be important factors in determining aggregate crime patterns because they bring offenders and victims together in space and time. Activity spaces and domains represent the areas with which an individual has routine interaction.

Research on offender travel behavior has shown that their activity spaces are similar to non-offenders but more diffuse, including approximately two blocks on either side of major arteries, entertainment and employment centers (Paul Brantingham & Brantingham, 1991). The more diffuse pattern is a result of additional exploration to identify potential targets. In addition, studies on the journey to crime have consistently found a distance decay effect (i.e., the distance from residence to crime location is shorter rather than longer) (Capone & Nichols, 1976; Costanzo, Halperin, & Gale, 1986; Groff & McEwen, 2005; Katzman, 1981; McIver, 1981; G. F. Rengert, Piquero, & Jones, 1999). Together, the principle of least effort and findings from distance decay and activity space research emphasize the importance of proximity, accessibility, land use, and individual characteristics in the convergence of victims and offenders. The formulation of activity spaces in this model combines elements from the perspectives just discussed and relies heavily on the notion of an integrated time-space approach to modeling human behavior in the urban environment.

## 2.3 Hypotheses

The review of the relevant literature highlights two questions related to the role of time and space in the production and pattern crime events. Does the amount of time spent away from home continue to impact crime in the predicted direction regardless of the spatio-temporal structure of routine activities? What effect do temporal and spatio-temporal constraints on routine activities have on the incidence and pattern of street robbery events? To examine these questions the current research revisits an earlier study's hypotheses ($H_1$ and $H_3$) (Groff, Forthcoming-a) and tests if they hold true when the structure of routine activities is varied. Two additional hypotheses ($H_2$ and $H_4$) are also tested. The first two hypotheses address the incidence of street robbery and the last two its spatial distribution:

> $H_1$: As the average time spent by civilians on activities away from home increases, the aggregate rate of robbery will increase.
>
> $H_2$: The temporal and spatio-temporal schedules of civilians while away from home change the incidence of robbery events.
>
> $H_3$: As the average time spent by civilians on activities away from home increases, the spatial pattern of robberies will change.
>
> $H_4$: The temporal and spatio-temporal schedules of civilians while away from home change the spatial pattern of robbery events.

The first hypothesis tests the core assertion of routine activity theory when agents have temporal and spatio-temporal schedules. The earlier study's findings demonstrated the plausibility of this hypothesis when agents had no constraints on their spatio-temporal when away from home (Groff, Forthcoming-a). The second hypothesis examines the effect of adding temporal and then spatio-temporally defined activity spaces on the incidence of street robbery. The third hypothesis is replicated from the original study and compares the outcome distributions of street robberies of the experimental conditions (i.e. as society spends more time away from home) to one another. The fourth hypothesis explores the impact of changing the structure of routine activities on the spatial distribution of crime events. Since this is the first test of the effect of spatio-temporal schedules on the spatial pattern of street robberies, hypotheses 3 and 4 do not describe the potential outcome pattern but simply note it will be different.

## *3.0 Research Design*

This section describes the overall methodology used to implement two additional versions of a basic model of street robbery. It begins with a brief overview of agent-based modeling in general and the software used to implement the model. Next, the input data utilized and the experiments that are conducted to test the effects of space and time on the frequency and distribution of crime events are discussed.

### 3.1 Agent-based Modeling and the Implementation Software

Agent-based models are one type of simulation modeling. An agent-based model consists of a collection of autonomous entities implemented within a software program (O'Sullivan & Haklay, 2000). Taken together these entities create an artificial world in which agents can represent people, governments, neighborhoods etc. Each entity has a set of unique characteristics and behaviors which are often based on existing theory and empirical research. Typically, these agents interact within an artificial world although there is increasing recognition of the value of using geographic information systems to provide a 'real' landscape (Brown et al., 2005; O'Sullivan & Haklay, 2000).

The two additional versions of the original model (Groff, Forthcoming-a) are built using the same software, Agent Analyst.[30] Agent Analyst combines two of the most popular packages for ABM and GIS. For ABM it relies on the Recursive Porous Agent Simulation Toolkit (Repast) product line and for GIS it uses ArcGIS. Once the Agent Analyst toolbox is added into an ArcGIS session individual models can access

[30] Agent Analyst is under development as a partnership between ESRI and Argonne National Laboratories; the parent companies of ArcGIS and Repast respectively. Agent Analyst is free but currently available by request only. The website for Repast is http://repast.sourceforge.net/.

shapefiles allowing: 1) individual agents to become spatially aware and 2) the visualization of agent movement and decision outcomes (e.g. locations of crimes).

A combined ABM/GIS simulation model integrates the advantages of autonomous agents found in agent-based modeling with the spatial explicitness of a geographic information system (Albrecht, 2005; An et al., 2005; Brown et al., 2005). Both are necessary to move from artificial environments to 'real' ones. The agents interact on city streets and their activities during the simulation are impacted by the distribution of opportunities for housing, employment, shopping and recreation across the urban backcloth. In this way, the spatial behavior of agents based on real landscapes is more representative of actual human behavior than that of agents who are created with and interact upon artificial landscapes. [31]

### 3.2 Data

The model uses input data describing the land use and street network of Seattle, Washington to provide the basis for the model landscape and the agent activity spaces (Groff, Forthcoming-a). Four datasets describing conditions in Seattle are used to inform the activity spaces of agents in the model: 1) total population; 2) total employment; 3) total potential activities; and 4) streets. Blockgroup level population figures describe the distribution of residences across Seattle (U.S. Census Bureau, 2000). Employment data provide the number of employees per zip code area (U.S. Census Bureau, 2002). Potential activity locations are quantified through the use of retail, recreation and service establishments (e.g. grocery stores, convenience stores, dry cleaners, gyms etc.) (ESRI, 2003). Finally, the King County Street Network

[31] The details of implementing movement and activity spaces in the model are discussed in Groff (Forthcoming-b).

Database (SND) file is used to structure the agent's movements. Because of software limitations, street intersections are used to represent places rather than streets. Individual civilians and police in the model move from street intersection/node to street intersection/node (hereafter referred to as node). There are 16,035 nodes in Seattle and these locations represent places at which a street robbery may occur.

In addition to the input data describing Seattle, twelve parameters are set prior to the model run (Table 4-1). These parameters are identical to the ones used in the previous study (Groff, Forthcoming-a). The choice of parameter values is a critical aspect of all models that deserves special attention because of the potential impacts on the model outcomes. Findings from the original model were robust to sensitivity tests (i.e. changes in both parameter values and random number seeds) (Groff, Forthcoming-a). The same sensitivity tests are conducted here: 1) five of the parameter values are increased; and 2) the random number seed is changed four times. The model runs are repeated for each test and a one-way ANOVA is applied to evaluate the results.

Table 4-1: Parameters in the Model

| Variable | Rationale |
|---|---|
| **Society Level** | |
| Number of Agents = 1,000 | Represents a balance between ensuring there are enough agents so that interactions can occur and the computational overhead from using more agents |
| Number of Cops = 200 | Chosen to ensure that cops would be present at some of the convergences that occur across the 16,035 places in Seattle. |
| Unemployment Rate = 6% | The unemployment rate of six percent is based on the 2002 unemployment rate for Seattle (Bureau of Labor Statistics, 2003).[32] |
| Rate of Criminal Propensity = 20% | Given that 20% of the population has committed a crime, 20% of civilians are assigned criminal propensity using a uniform distribution (Visher & Roth, 1986). |
| Time To ReOffend = 60 | Parameter value chosen as a starting point since the author could find no empirical data on which to base time to reoffend.. |
| Random Number Seed = 100 (seed also tested at 200, 300, 400 and 500) | An explicit random number seed based on the Mersenne Twister (MT) algorithm is used as the basis for all random number distributions used in the model. MT is currently considered to be the most robust in the industry (Ropella et al., 2002). |
| **Agent Level** | |
| Societal Time Spent Away From Home = 30% (40%, 50%, 60%, 70%) | Assigned based on a normal distribution with a mean of 432 minutes (for the 30% condition) and a standard deviation of 10% of the mean (sd = 43).[33] |
| Initial Wealth = 50 | Initial wealth is assigned with a mean of 50 and a standard deviation of 20 units. |
| Amount of wealth received each payday = 5 | No empirical evidence available. |
| Amount of wealth exchanged during robbery=1 | No empirical evidence available.[34] |
| **Situation Level** | |
| Guardianship Perception = U(-2,2) | The guardianship perception value can add or subtract zero, one or two guardians from the actual number present. This represents the stochastic element in the offender's perception of the willingness of a guardian to intervene. |
| Suitable Target Perception = U(-1,1) | The value in suitable target can increase or decrease the suitability or leave it unchanged. This enables the offender to sometimes decide a target is not suitable even when they have more wealth. |

[32] Since the jobs data are from 2002, the corresponding year's unemployment rate is used.
[33]The time spent away from home is systematically varied to test the core proposition of routine activity that as time spent away from home increases crime will increase.
[34] A request to the Seattle Police Department for the average amount of cash taken during street robberies remains unanswered.

## 3.3 Agent Activity Spaces in the Model

The purpose of the agent activity spaces in these versions of the model is two-fold. First, they represent the *routine activities* element of RAT. Second, they provide a systematic way of testing the impact of time and time-space schedules on the incidence and distribution of street robbery. While simple, the initial definition of activity spaces captures some crucial elements of human spatial behavior. First, agents like people, have a series of places that they visit each day representing home, work and activities. For instance, a person may go to work, pick-up the dry-cleaning, go to the gym and then to the grocery. Second, the spatial extent of those activities is fairly consistent and forms a routine activity space. One limitation of the activity spaces in the model is that they are static while human behavior often changes daily. Ideally, each the agent's activity space would be dynamic during the model run enabling them to choose locations for activities within their potential path area given existing temporal constraints. However, current software limitations preclude dynamic activity spaces. While the software limitation is disappointing, the implementation of routine activity spaces with temporal and spatio-temporal constraints represents an advance and is tested here.

A time-geographic framework is supplemented by work done on activity spaces to develop simulated routine activities (Kwan & Lee, 2004). This hybrid approach gives structure to the way human activities are organized, while recognizing the importance of activity spaces as the areas in which individuals conduct their everyday lives (Horton & Reynolds, 1971). Following Hägerstrand (1975), the activity spaces developed here are not meant to accurately portray the specific activities, durations and routes of each individual in the model or to predict what those activity spaces would look like but rather to offer a representative heuristic. In this way, the current model looks for middle-ground between empirical research and simulation which more effectively captures the complexity of human behavior.

As in real life, activity spaces in the model have both spatial and temporal elements. Theory holds that the travel behavior of individuals is influenced by the: 1) street network; 2) the specific locations at which opportunities for employment, recreation, retail and services exist; and 3) the distance among those locations (Hägerstrand, 1970;1975; Kwan, 1998; H. J. Miller, 1991). The temporal schedule is impacted by the amount of time to spend at each activity, the distances between the activities and the speed of travel. The more time spent traveling the less is available to spend at an activity.

In the model, the routine activity spaces of individuals are implemented as a set of nodes (places) and paths (list of places traversed when traveling from one node to another) (Hägerstrand, 1970;1975; H. J. Miller, 1991). Specifically, each civilian agent is assigned four nodes representing a home, a main (e.g. work, school etc.) and two activities (e.g. recreation, social, and retail places). The nodes are assigned based on the distributions of population, jobs and activities in Seattle (e.g. if 10% of the population lives in a particular blockgroup then 10% of the agents are assigned to that blockgroup). In this way, the size and form of activity spaces is influenced by the distribution of residential housing, jobs, schools, retail and services (Kwan, 1998; Weber & Kwan, 2003). The outcome is that each civilian has a unique activity space reflecting their origin/end place, the places they visit, and the routes among those places.[35]

The agent activities are attached to a series of street intersections rather than street addresses or street blocks.[36] Street intersections are used to represent "interchange points for journeys from stations to the road network and *vice versa*" (Lenntorp, 1978 168). It is at street intersections that agents change travel status (e.g. from work to travel) and that the potential for street robbery exists. Intersections also provide a convenient heuristic to represent the dual nature of many places; a bank may be a workplace to an employee and a discretionary activity to customers (Lenntorp, 1978).

In addition, domains can change as people change jobs or as circumstances change (Hägerstrand, 1970). Accordingly, each civilian agent has two potential activity spaces; one activity space is used while employed and the other while unemployed. Since becoming unemployed does not automatically change residence and other activities, the two activity spaces are identical except that the work location is dropped from the unemployed path and a new activity location is added. The home location and the locations of the original two activities do not change.

Following time-geographic principles each agent is indivisible, they can be in only one place at a time, movement across space takes time, and all activities have duration. These principles are reflected in the existence of a temporal schedule for each that incorporates the temporal constraints on their travel (H. J. Miller, 2005).

[35] This relatively simple representation of human spatial behavior does not incorporate other aspects of trip decision making (e.g. trip purpose, model of travel, order of travel, time of day etc.) which affect travel but it does provide a starting point.
[36] This follows the method used by Miller (1991) but for different reasons. His was to simplify the representation in GIS. Here the use of street intersection reflects a software limitation. Repast cannot read network or geodatabase files from ArcGIS.

Each agent's temporal schedule is primarily based on the daily amount of time the agent is assigned to stay at home and the size of their activity space. The amount of time to stay at home is assigned first and is static so that the average time spent away from home for the societal experimental condition will be accurate. Next, the amount of time needed to travel among the activity nodes is calculated since, as noted earlier, it limits the time available to spend at activities. The remaining time in the day is randomly allocated to the Main, Activity 1 and Activity 2.

The final element important to human activity is the street network. The paths taken to travel among activity place are structured by the street network (Hägerstrand, 1970;1975; H. J. Miller, 1991). Movement in the model is either random or directed. Random movement follows a 'random walk' process in which a node is randomly chosen from the set of adjacent nodes (Chaitin, 1990). Random movement is used by the police agents in all three versions and by the civilians in two versions of the model.

Agents with pre-defined spatio-temporal activity spaces (i.e. those in the Activity Space version) are the only ones who use directed movement. Their activity locations and the path among those places are generated in ArcGIS before the start of the model. Each agent starts at home and then moves along the shortest path from activity place to activity place according to a temporal schedule and following a ring pattern so they end at home (Hägerstrand, 1970; Lenntorp, 1978; H. J. Miller, 1991). While research has shown that the shortest topological path is frequently not taken, it offers a standardized and convenient heuristic for this initial model. In this way, movement along the street network and activity spaces provide the basis for modeling how the routine nature of spatio-temporal behavior influences the convergence of individuals at a place-time.

## 3.4 Experiments

In keeping with the original study, the same series of five experiments are conducted on each of the two new versions of the street robbery model (Table 4-2). These experiments are used to test: 1) whether changes in routine activities (defined as time spent away from home) can increase crime even if the numbers of motivated offenders remains constant; and 2) the impact of spatial and temporal constraints on the incidence and spatial pattern of street robberies. These tests proceed in a systematic fashion, with each condition representing an increase in the societal average for time spent on routine activities away from the home. All of the percentages represent an average time spent away from home for the agent population as a whole; individual agents have different times spent away from home.

Table 4-2: Experimental Conditions: Three Versions

| **Version of Model** | **Average Time Spent Away From Home** | | | | |
|---|---|---|---|---|---|
| | **C1** | **C2** | **C3** | **C4** | **C5** |
| **Simple** | 30% | 40% | 50% | 60% | 70% |
| **Temporal** | 30% | 40% | 50% | 60% | 70% |
| **Activity Space** | 30% | 40% | 50% | 60% | 70% |
| Hours per week | ≈50 | ≈67 | ≈84 | ≈101 | ≈118 |

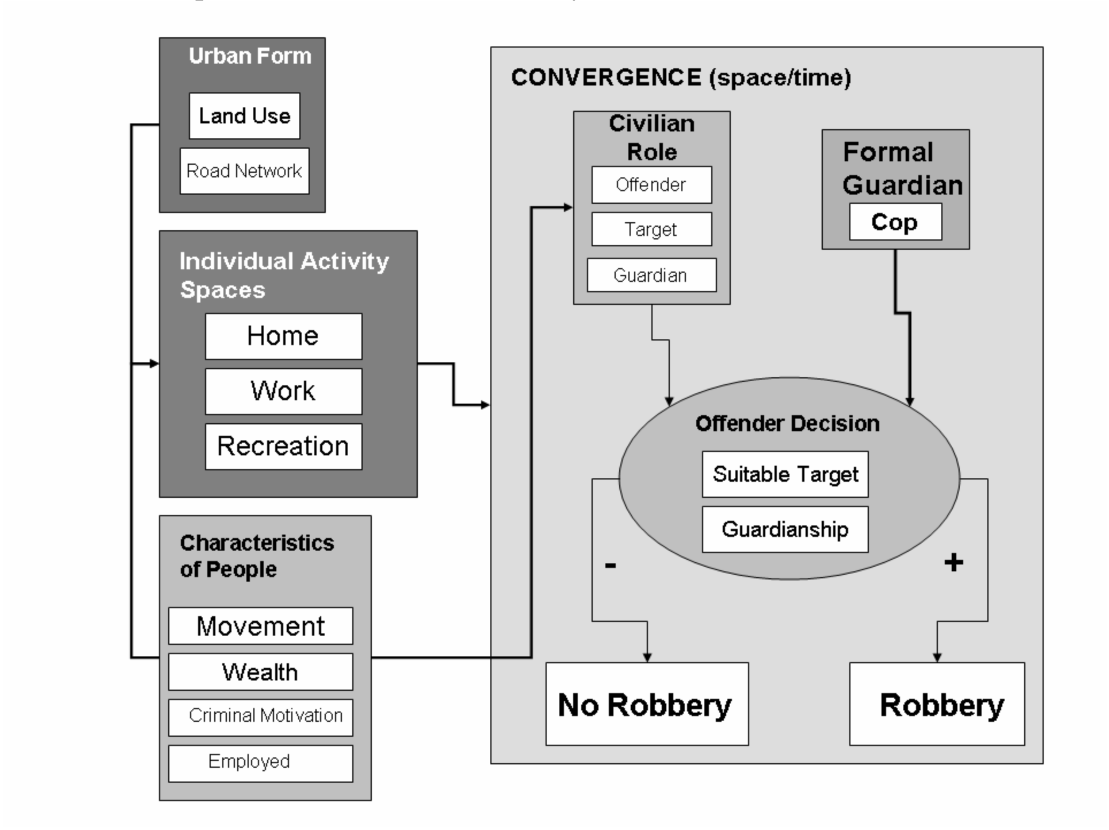
Random number generators play an important dual role in agent-based models by providing the stochastic elements in the model and enabling scientific experimentation. Both uniform and normal random number distributions are used for decision-making in the model. For example, random numbers play a key role in representing uncertainty in the current knowledge about how individuals evaluate guardianship and target suitability. However, when a random number seed is defined at the start of a simulation the random number generator produces the same sequence of random numbers each time the model is run making experiments repeatable. This characteristic forms the basis for using simulation as a laboratory for experimentation because it enables any differences in the outcome variable to be attributed to the manipulated variable and not to other sources (Axelrod, Forthcoming; Groff, Forthcoming-a; Lenntorp, 1978).

## *4.0 Implementation Model*

This section explains how the original model of street robbery is extended by creating two additional and progressively more complex versions. Like the original model of street robbery, these versions are based on the core elements of RAT but add routine activity spaces (Figure 4-1). Since the original model serves as a base for the two new versions, the elements common to all three versions are described first and then the changes unique to each additional version are covered.

There are two types of people in the model, civilians and police. The civilians represent the general population of Seattle. Only civilians have attributes and can take on different roles in the model (i.e., offender, victim, or guardian) depending on the particular situation. Each civilian in the model has a unique set of characteristics that include criminal propensity, time to stay at home, wealth, and employment status.

Figure 4-1: Conceptual Model of Street Robbery for All Versions



Criminal propensity differentiates agents who are interested in committing robbery from all other agents in the model. Civilians with criminal propensity can potentially take on any one of three roles, offender, victim, or guardian. Civilians without criminal propensity can be either victims or guardians. In all other ways, civilians with criminal propensity are exactly the same as those without. While only agents with criminal propensity can make the decision to offend, it is the particular constellation of individual and contextual dynamics that determines whether a crime is committed. In this way, patterns of offending and victimization are allowed to emerge from decisions made by individuals in particular contexts. In addition to criminal propensity civilian agents are assigned a time to spend at home that is static over a model run, and a wealth and employment status that can change during the model run. Once convergence occurs, guardianship and suitability of target are considered by the offender when making the decision whether or not to commit a robbery.

Police agents represent formal guardian and their presence automatically prevents a crime from occurring. In all model versions, at the start of the simulation police agents are randomly distributed across the nodes and they follow a 'random walk' movement pattern in which they move one node at a time and only to an adjacent node.

Regardless of model version, the decision to offend is made as follows. At each model tick (i.e. each minute of the model year) all nodes with at least one agent present are evaluated. Active nodes meeting the following three criteria are evaluated further: 1) no police present; 2) at least two civilians present; and 3) at least one of the civilians must have criminal propensity. If there is only one offender at the node, that agent automatically becomes the active offender. Otherwise, the active offender is randomly selected from the list of agents with criminal propensity who are at the node. Offender agents who are not selected to be active are at risk of becoming victims. When an agent commits a robbery, one unit of wealth is taken from the victim and transferred to the offender. Once the active offender at each of the active nodes evaluates their situation, all agents move and the decision structure repeats.

### 4.1 Model Versions

Each of the three street robbery model versions is described and the differences among them are highlighted (Table 4-3). The Simple version is from the original study and is included to offer a counterpoint to the two new versions (Groff, Forthcoming-a). In the Simple version, the agents move randomly along a real street network but the version does not incorporate the notion of temporal or spatio-temporal schedules. Civilian agents are randomly distributed across the nodes and each day begins at the node where the previous one ended. Since they are 'at risk' of being robbed or robbing whenever they are not at home, civilians in the Simple version are the most exposed to becoming victims of or committing a crime. Unlike the other two versions, civilian agents do not have an employment status so all civilian agents get paid every other week.

Table 4-3: Implementation Versions of the Conceptual Street Robbery Model

| | **Simple** | **Temporal** | **Activity Space** |
|---|---|---|---|
| **Civilian Movement** | Random | Random | Defined Activity Space |
| **Police Movement** | Random | Random | Random |
| **Civilian Characteristics** | | | |
| Criminal Propensity | Yes | Yes | Yes |
| Wealth | Yes | Yes | Yes |
| Activity Space | No | Temporal Only | Spatio-temporal |
| Multi-faceted Risk Status | No | Yes | Yes |
| Employment Status | No | Yes | Yes |

The Temporal and Activity Space versions are very similar to one another differing only in that one has temporal constraints and the other has spatio-temporal constraints on agent behavior. The Temporal version has civilian agents who are randomly distributed across the nodes and follow a temporal schedule for travel when not at home. Civilians in the Temporal and Activity Space models share the same temporal schedule for activities and travel and consequently those agents spend the same amount of time 'at risk' for street robbery. The Activity Space version of the model implements geographically informed activity spaces in which each agent travels among a pre-defined set of activity nodes.

Civilian agents in the Temporal and Activity Space versions have attributes describing the time to spend at home, at a main activity, and at two other activities. Each type of activity has a different level of risk for street robbery. While at home or at work the agent is not at risk of participating in a street robbery. Thus, the amount of time at home or at work reduces risk of street robbery, while time spent traveling or engaging in other activities increases it. This representation of risk is in keeping with the crime being studied. By definition, street robbery happens only on the street or in public places; not in a home or inside a workplace.[37]

Civilian agents in the Temporal and Activity Space versions of the model each have an employment status. This characteristic has two important impacts in these versions of the model. First, it changes the amount of time spent at the three activity nodes (but not the overall time spent away from home). In the case of the Activity Space version, employment status determines the spatial-temporal aspects of the agent's activities. Those agents who are employed receive regular but static infusion of wealth every two weeks over the model year but civilians who are unemployed do not get paid. Every month, 3% of unemployed agents become employed and are replaced by a new random selection of employed agents who become unemployed. It

[37] The designation of 'at risk' is simplified from real life. A person who is shopping in a retail store also cannot be a victim of street robbery but is considered 'at risk' in the model. The main purpose of the designation is to vary the time a civilian agent is at risk based on their activities.

is important to note that the employment status is assigned independently of the criminal propensity indicator; civilians with criminal propensity can be employed in the model, as they are in life.

## *5.0 Analysis*

Descriptive statistics such as mean, median, standard deviation, and range are employed to characterize the results of each of the experiments and to compare them. An ANOVA, is used to determine if there is a significant difference among the street robbery rates for the five experimental conditions across the model versions (Axelrod, Forthcoming). The response variable is number of robberies for each of the one thousand civilian agents.

Two approaches for describing the spatial distribution of street robberies are applied (Groff, Forthcoming-a). A visual comparison of the resulting crime patterns is made using a kernel density. Kernel density surfaces offer a means of evaluating the existence of global trends in the distribution of street robberies and for comparing the relative density of robberies across experimental conditions (Bailey & Gatrell, 1995; Levine, 2005; Mitchell, 1999; Williamson et al., 2001).

In addition, formal tests of the spatial distribution of crime events are employed via Ripley's *K* but with the following interpretational caveat in mind. As noted in the earlier study, the data from the model are somewhat unique in that convergences and robbery events can only occur at the street nodes thus a comparison to complete spatial randomness is inappropriate (Forthcoming-a). Instead, the *K* function is calculated for the street nodes and then compared to CSR. The original study found the street nodes were significantly more clustered than would be expected under CSR.

In addition, the robberies were more clustered than the street nodes at certain distances. To investigate this aspect in terms of space and time, both the line for CSR and the *K* function are depicted on all graphics to reveal the degree of intrinsic clustering in the street network as compared to CSR.

The impact of schedule constraints on the spatial distribution of street robbery is evaluated by comparing the findings from the Simple version to those from the Temporal and Activity Space versions across the five experimental conditions using kernel density and Ripley's *K*. All of these measures are distance-based and characterize the spatial patterning of the street robbery locations as conditioned by the pattern of street nodes. The robustness of all the findings is then evaluated through the systematic variation of five of the key parameter values and by varying the random number seed used across the new versions of the model.

## *6.0 Findings*

This section summarizes the findings of the analyses just described. First descriptive model outcomes are expressed by examining both place- and societal-level attributes to characterize differences in the results from the model runs of the three versions across the five experimental conditions. Next, the results pertinent to each of the hypotheses are conveyed.

### 6.1 Descriptive Analysis

Societal-level changes in the number of street robberies, the frequency of convergence of agents in space-time (i.e. opportunities for street robbery), and the number of crimes deterred by the police for all three versions of the model are in line with what routine activity theory would expect; all the values increase steadily with time spent away from home (Table 4-4). While the overall trends are consistent, significant differences in volume exist by model version. The Simple model has the highest number of robberies and the steepest increases as time spent away from home increases. The Temporal version has the fewest robberies and an identical slope as the Activity Space version. Together these results point to the importance of a time schedule in lowering the incidence of street robberies regardless of the time spent away from home. The addition of a spatially-constrained schedule to the Temporal version increases the number of street robberies. This outcome is most likely related to the rate of convergence (i.e. presence of motivated offender and suitable target at same place-time) which tends to be highest in the Activity Space version and lowest in the Temporal version.

Table 4-4: Societal-level Model Outcomes

| | **Experimental Condition** | | | | |
|---|---|---|---|---|---|
| | **30%** | **40%** | **50%** | **60%** | **70%** |
| Target time to spend away from home in minutes (hours) | 432<br>(7.2) | 576<br>(9.6) | 720<br>(12) | 864<br>(14.4) | 1008<br>(16.8) |
| Actual time spent away from home | 436.9 (S)<br>427.8 (T)<br>427.7 (AS) | 580.2 (S)<br>572.5 (T)<br>572.3 (AS) | 723.5 (S)<br>717.0 (T)<br>716.9 (AS) | 866.8 (S)<br>861.6 (T)<br>861.5 (AS) | 1010.1 (S)<br>1006.2 (T)<br>1006.2 (AS) |
| | | | | | |
| Total Robberies | 54,637 (S)<br>12,807 (T)<br>32,326 (AS) | 76,032 (S)<br>13,671 (T)<br>34,628 (AS) | 95,219 (S)<br>15,183 (T)<br>38,331 (AS) | 118,085 (S)<br>16,196 (T)<br>41,266 (AS) | 139,007 (S)<br>17,181 (T)<br>46,085 (AS) |
| Total Convergences | 1,454,341 (S)<br>736,787 (T)<br>1,889,899 (AS) | 2,050,761 (S)<br>1,013,814 (T)<br>2,663,961 (AS) | 2,631,149 (S)<br>1,285,568 (T)<br>3,446,132 (AS) | 3,238,760 (S)<br>1,579,963 (T)<br>4,260,133 (AS) | 3,835,299 (S)<br>1,880,647 (T)<br>5,018,754 (AS) |
| Total Robberies Deterred by Police | 1,532 (S)<br>325 (T)<br>1,286 (AS) | 2,148 (S)<br>414 (T)<br>1,417 (AS) | 2,693 (S)<br>416 (T)<br>1,484 (AS) | 3,430 (S)<br>454 (T)<br>1,670 (AS) | 4,040 (S)<br>450 (T)<br>1,979 (AS) |
| Percentage of civilians who were robbed | 77.7% (S)<br>74.5% (T)<br>74.0% (AS) | 77.6% (S)<br>73.2% (T)<br>72.8% (AS) | 76.4% (S)<br>74.6% (T)<br>71.8% (AS) | 75.1% (S)<br>72.5% (T)<br>72.5% (AS) | 76.2% (S)<br>71.5% (T)<br>73.5% (AS) |
| Percentage of civilians who were repeat victims of street robbery | 65.2% (S)<br>64.6% (T)<br>64.4% (AS) | 65.2% (S)<br>64.1% (T)<br>63.1% (AS) | 65.5% (S)<br>63.9% (T)<br>62.6% (AS) | 65.1% (S)<br>63.2% (T)<br>62.9% (AS) | 65.7% (S)<br>62.6% (T)<br>63.4% (AS) |
| Number of civilians with criminal propensity who committed a street robbery | 200 (S)<br>199 (T)<br>200 (AS) | 200 (S)<br>200 (T)<br>200 (AS) | 200 (S)<br>200 (T)<br>199 (AS) | 200 (S)<br>200 (T)<br>198 (AS) | 200 (S)<br>200 (T)<br>197 (AS) |

(S) Simple (T) Temporal (AS) Activity Space

Results from this research also show that deterrence (i.e. number of times the presence of a police agent prevents a robbery from taking place) increases for all model versions as the societal time spent away from home increases and the relationships among the versions are identical to those for convergence. This supports Cohen and Felson's (1979) hypothesis that the frequency of convergence impacts the potential for deterrence. Whenever there are more convergences there are by definition more times a police agent can function as an agent of formal guardianship.

Together these findings illustrate the separate impact of a temporal schedule and a defined activity space on the frequency of convergences across the three models. Agents who travel randomly but have a temporal schedule experience the fewest number of convergences because the time they are 'at risk' is less then the agents in the Simple version. When a spatial element is added (i.e. agents have defined activity spaces), it increases the frequency of convergence because agent's homes, jobs and activities are clustered as opposed to randomly allocated across Seattle as in the other versions. Increasing convergence translates into more street robberies for agents in the Activity Space version.

## 6.2 Hypothesis Test Results

The first hypothesis tests the core assumption of RAT; *as the average time spent by civilians on activities away from home increases, the aggregate rate of robbery will increase*. A One-Way ANOVA is applied to the means of the five experimental conditions to determine if the average number of robberies across all the civilian agents increases as the time spent away from home increases. Separate tests are conducted for the Temporal and Activity Space versions of the model. The results of the ANOVA indicate significant differences only for the Temporal version (Table 4-5).[38] Overall, the only version that does not support RAT is the one that includes space.[39]

Table 4-5: ANOVA for Street Robbery Events across Versions and Experimental Conditions

| | Proportion of Time Spent Away From Home | | | | |
|---|---|---|---|---|---|
| | Condition 1 (30%) | Condition 2 (40%) | Condition 3 (50%) | Condition 4 (60%) | Condition 5 (70%) |
| **Number of civilians** | N=1,000 | N=1,000 | N=1,000 | N=1,000 | N=1,000 |
| | | | | | |
| **Simple Model ***** | | | | | |
| Mean (Standard Deviation) | 54.64 (101.99) | 76.03 (144.15) | 95.22 (182.35) | 118.09 (228.14) | 139.01 (270.06) |
| **Temporal Model ***** | | | | | |
| Mean (Standard Deviation) | 12.81 (17.54) | 13.67 (19.35) | 15.18 (22.42) | 16.20 (24.34) | 17.18 (26.64) |
| **Activity Space Model** | | | | | |
| Mean (Standard Deviation) | 32.33 (87.69) | 34.63 (103.26) | 38.33 (129.42) | 41.27 (148.5) | 46.09 (174.90) |

*** Difference among one or more of the groups is significant at P <= .000.

Additional tests using Tamhane's T2 are employed to identify which groups differed significantly (Table 4-6).[40] Comparing each group, in turn, to the other four groups reveals that there are differences between the conditions in the Simple and Temporal

[38] Because of the positive skew to the distribution of robberies, additional tests regarding the equality of means are conducted. Both the Brown-Forsythe and the Welch tests for equality of the means are significant at .000. These tests are preferable to the F test when the equality of variances assumption is violated as it is here ("SPSS for Windows," 2002).

[39] The large sample size has a twin effect producing both a powerful design capable of detecting even small effects and making finding statistical significant relationshipts more likely. The non significant finding for the Activity Space version may stem from the large standard deviation found in each of the conditions. A variety of additional analysis confirm the finding of non significance for the Activity Space model. Specifically, analyses conducted with the same response variable under a Univariate Generalized Linear Model (GLM), and additional tests using both a One-way ANOVA and Univariate GLM but with a logged response variable, produce consistent findings.

[40] Tamhane's T2 is used because it does not assume equal variances. A test for homoscedasticity showed the variances are not equal across the five experimental conditions. The Levene statistic is significant indicating the variances are significantly different among the groups. However, ANOVA is robust in the face of this violation when the group sizes are equal which they are in this research (Newton & Rudestam, 1999; Shannon & Davenport, 2001).

versions.[41] While all but two comparisons for the Simple version were significant, only three of the between-group differences are significant for the Temporal version; between the 30% and both the 60% and 70% conditions as well as between the 40% and the 70% condition. Thus, the effect of a temporal activity schedule is to reduce the number of significant differences between the experimental conditions.

[41]Tamhane's T2 is only applied to those versions in which there were significant differences for the version as a whole.

Table 4-6: Post Hoc Tests of Mean Differences by Experimental Condition (seed = 100)

| *(I) Randomization Condition* | *(J) Randomization Condition* | *Mean difference (I - J)* | *Standard error* | *Significance* |
|---|---|---|---|---|
| 30% Time away | 40% Time away | | | |
| | (S) [a] | -21.39 | 5.584 | .001 |
| | (T) | -.86 | .826 | .970 |
| | (AS) | -2.30 | 4.284 | 1.00 |
| | 50% Time away | | | |
| | (S) [a] | -40.58 | 6.607 | .000 |
| | (T) | -2.38 | .900 | .081 |
| | (AS) | -6.01 | 4.944 | -6.01 |
| | 60% Time away | | | |
| | (S) [a] | -63.45 | 7.903 | .000 |
| | (T) [a] | -3.39 | .949 | .004 |
| | (AS) | -8.94 | 5.453 | .656 |
| | 70% Time away | | | |
| | (S) [a] | -84.37 | 9.129 | .000 |
| | (T) [a] | -4.37 | 1.009 | .000 |
| | (AS) | -13.76 | 6.187 | .234 |
| 40% Time away | 50% Time away | | | |
| | (S) | -19.19 | 7.351 | .088 |
| | (T) | -.51 | .936 | .676 |
| | (AS) | -3.70 | 5.236 | .999 |
| | 60% Time away | | | |
| | (S) [a] | -42.05 | 8.534 | .000 |
| | (T) | -2.53 | .983 | .098 |
| | (AS) | -6.64 | 5.719 | .941 |
| | 70% Time away | | | |
| | (S) [a] | -62.98 | 9.681 | .000 |
| | (T) [a] | -3.51 | 1.041 | .008 |
| | (AS) | -11.46 | 6.423 | .540 |
| 50% Time away | 60% Time away | | | |
| | (S) | -22.87 | 9.236 | .126 |
| | (T) | -1.01 | 1.046 | .983 |
| | (AS) | -2.93 | 6.229 | 1.00 |
| | 70% Time away | | | |
| | (S) [a] | -43.79 | 10.305 | .000 |
| | (T) | -2.00 | 1.101 | .515 |
| | (AS) | -7.75 | 6.880 | .951 |
| 60% Time away | 70% Time away | | | |
| | (S) | -20.92 | 11.180 | .470 |
| | (T) | -.98 | 1.141 | .993 |
| | (AS) | -4.82 | 7.255 | .999 |

[a] Significant differences were found between experimental conditions *I* and *J* at $p < .05$.

The second hypothesis is that *the temporal and spatio-temporal schedules of civilians while away from home change the incidence of robbery events*. This hypothesis explores whether the versions of the model produce significantly different numbers of street robberies for each of the experimental conditions (e.g. whether the number of robberies under the 30% time away condition for the Simple version were significantly different than under the Temporal or the Activity Space versions). The results of the ANOVA indicate there are significant differences between the rates of street robbery for all three versions of the model. A post hoc analysis reveals there are significant differences among all five experimental conditions (Table 4-7). In other words, regardless of the amount of time spent away from home, including the temporal and spatial components of activity spaces resulted in significantly different robbery rates. These results support the separate importance of both time and space when modeling routine activities.

Table 4-7: Post Hoc Tests of Mean Differences Between Same Condition in Different Model Versions (Seed = 100)

| *(I) Version* | *(J) Version* | *Mean difference (I - J)* | *Standard error* | *Significance* |
|---|---|---|---|---|
| Simple | Temporal | | | |
| | 30% Time Away [a] | 41.83 | 3.272 | .000 |
| | 40% Time away [a] | 62.36 | 4.599 | .000 |
| | 50% Time away [a] | 80.84 | 5.810 | .000 |
| | 60% Time away [a] | 101.89 | 7.255 | .000 |
| | 70% Time away [a] | 121.83 | 8.582 | .000 |
| | Activity Space | | | |
| | 30% Time Away [a] | 22.31 | 4.253 | .000 |
| | 40% Time away [a] | 41.40 | 5.607 | .000 |
| | 50% Time away [a] | 56.89 | 7.071 | .000 |
| | 60% Time away [a] | 76.82 | 8.608 | .000 |
| | 70% Time away [a] | 92.92 | 10.175 | .000 |
| Temporal | Activity Space | | | |
| | 30% Time Away [a] | -19.52 | 2.828 | .000 |
| | 40% Time away [a] | -20.96 | 3.322 | .000 |
| | 50% Time away [a] | -23.15 | 4.154 | .000 |
| | 60% Time away [a] | -25.07 | 4.758 | .000 |
| | 70% Time away [a] | -28.90 | 5.594 | .000 |

[a] Significant differences were found between model versions *I* and *J* at $p < .05$.

A tabular view of the spatial distribution of agent movement and robberies offers descriptive evidence in support of the third hypothesis; *as the average time spent by civilians on activities away from home increases, the spatial pattern of robberies will change* (Table 4-8). The outcome measures reveal both increases in concentration and spread of street robberies as time spent away from home increases. Mean robberies per node are lowest in the Temporal model followed by the Activity Space model. The more time that is spent away from home, the bigger the disparity among the versions. A different pattern emerges when looking at the percent of street nodes with only one robbery and the percent with more then one robbery. The Simple version has the highest proportions of both, followed by the Temporal version. Less than 10% of all nodes in Activity Space version ever have a robbery.

Table 4-8: Place-Level Model Outcomes

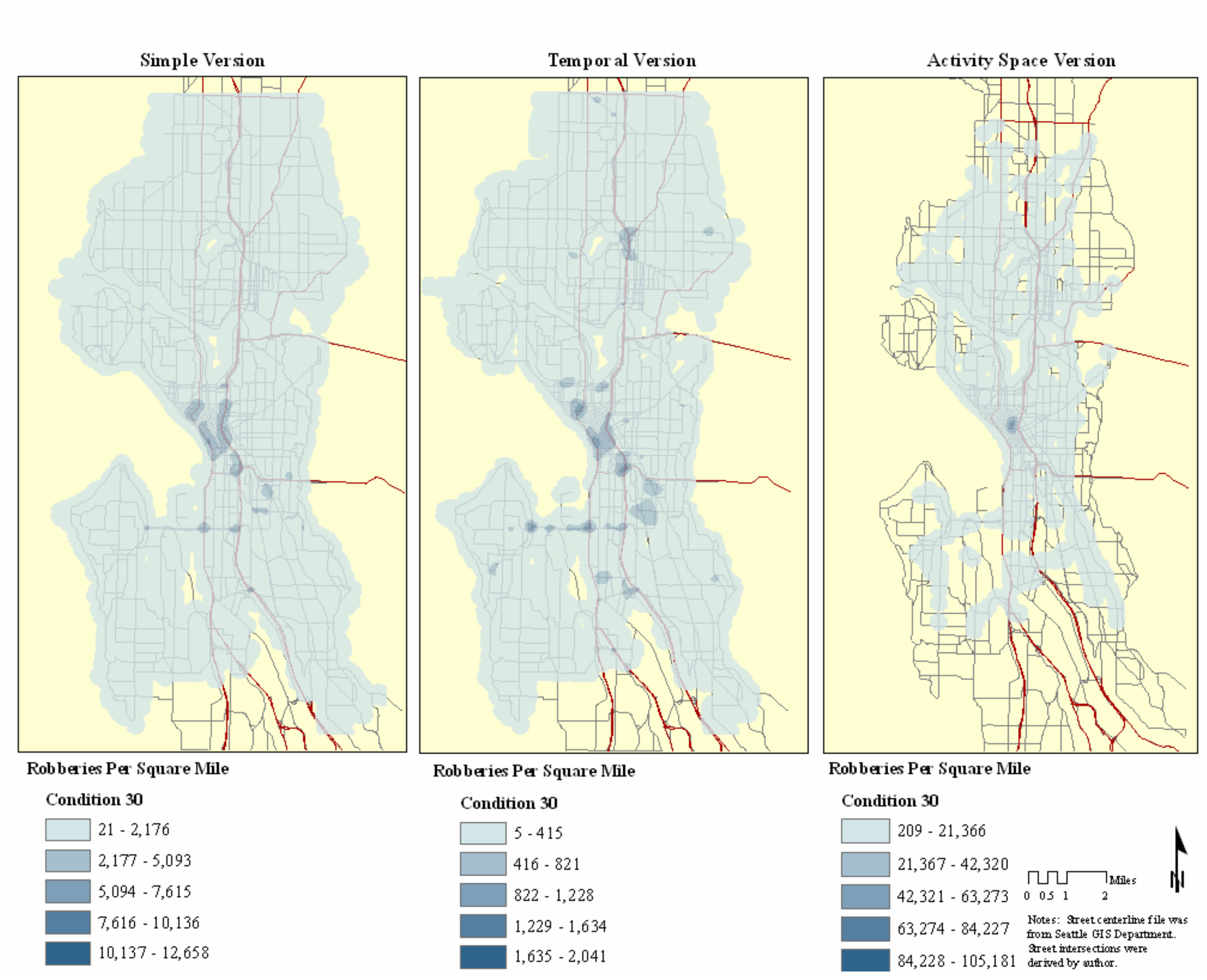
| | Experimental Condition | | | | |
|---|---|---|---|---|---|
| | 30% | 40% | 50% | 60% | 70% |
| Average robberies per node | 3.41 (S)<br>.80 (T)<br>2.02 (AS) | 4.74 (S)<br>.85 (T)<br>2.16 (AS) | 5.94 (S)<br>.95 (T)<br>2.39 (AS) | 7.36 (S)<br>1.01 (T)<br>2.57 (AS) | 8.67 (S)<br>1.07 (T)<br>2.87 (AS) |
| Average number of visits per node | 9,732.8 (S)<br>5,319.1 (T)<br>5,116.6 (AS) | 12,903.9 (S)<br>6,956.6 (T)<br>6,711.3 (AS) | 16,087.8 (S)<br>8,595.7 (T)<br>8,331.1 (AS) | 19,253.4 (S)<br>10,234.7 (T)<br>9,930.4 (AS) | 22,423.0 (S)<br>11,873.2 (T)<br>11,522.2 (AS) |
| Total places with a robbery | 13,376 (S)<br>6,689 (T)<br>1,535 (AS) | 13,925 (S)<br>6,641 (T)<br>1,499 (AS) | 14,309 (S)<br>6,549 (T)<br>1,472 (AS) | 14,531 (S)<br>6,441 (T)<br>1,475(AS) | 14,683 (S)<br>6,568 (T)<br>1,498 (AS) |
| Percent of places with a robbery | 83.4% (S)<br>41.7% (T)<br>9.6% (AS) | 86.8% (S)<br>41.4% (T)<br>9.4% (AS) | 89.2% (S)<br>40.8% (T)<br>9.2% (AS) | 90.6% (S)<br>40.2% (T)<br>9.2% (AS) | 91.6% (S)<br>41.0% (T)<br>9.3% (AS) |
| Total places with more then one robbery | 11,157 (S)<br>3,130 (T)<br>1,179 (AS) | 12,175 (S)<br>3,183 (T)<br>1,171 (AS) | 12,995 (S)<br>3,225 (T)<br>1,128 (AS) | 13,303 (S)<br>3,254 (T)<br>1,134 (AS) | 13,572 (S)<br>3,197 (T)<br>1,142 (AS) |
| Percent of places with more then one robbery | 69.6% (S)<br>19.5% (T)<br>7.4% (AS) | 75.9% (S)<br>19.9% (T)<br>7.3% (AS) | 81.0% (S)<br>20.1% (T)<br>7.0% (AS) | 83.0% (S)<br>20.3% (T)<br>7.1% (AS) | 84.6% (S)<br>19.9% (T)<br>7.1% (AS) |

(S) Simple (T) Temporal (AS) Activity Space

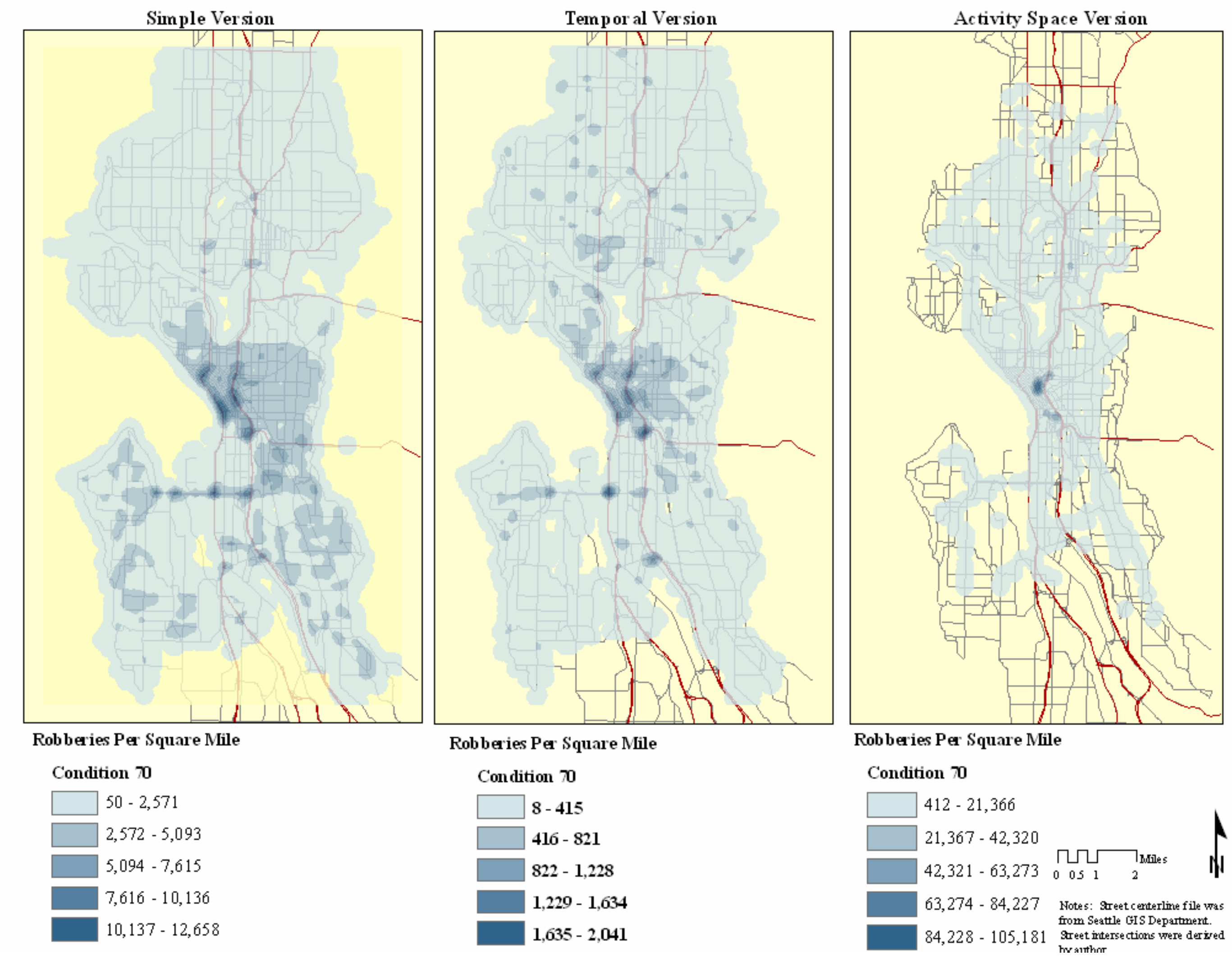
Another way of characterizing the spatial pattern of robberies is via a kernel density.[42] Two maps describe the spatial pattern of robberies that emerges for each version of the model at the 30 percent and 70 percent conditions (Maps 4-1 and 4-2). As civilians spend more time away from home, the robbery concentration at existing places increases while new areas emerge; an effect that is most likely due to the increased frequency of the convergence of the elements necessary for a crime to occur.[43] This visual inspection of the map series indicates support for the third hypothesis and illustrates the importance of considering the spatio-temporal structure of routine activities.

[42] The purpose of the kernel density surface use here is to represent the overall changes in intensity across the city of Seattle. Therefore, a bandwidth of 1,320 feet (one quarter mile) and a cell size of 100 feet are the basis for all kernel density surfaces. The quarter mile distance is often employed to represent the potential walking area for individuals in urban areas and by extension their potential area of interaction around a given point (Calthrope, 1993; Duaney & Plater-Zyberk, 1993; Nelessen, 1994). The surfaces are generated in ArcGIS version 9.1 and the output is in robberies per square mile (Mitchell, 1999).
[43] Kernel density maps of the 30, 40, 50 and 60 percent conditions are not shown here but are available from the author.

Map 4-1: Kernel Density for 30% Time Spent Away From Home



Map 4-2: Kernel Density for 70% Time Spent Away From Home



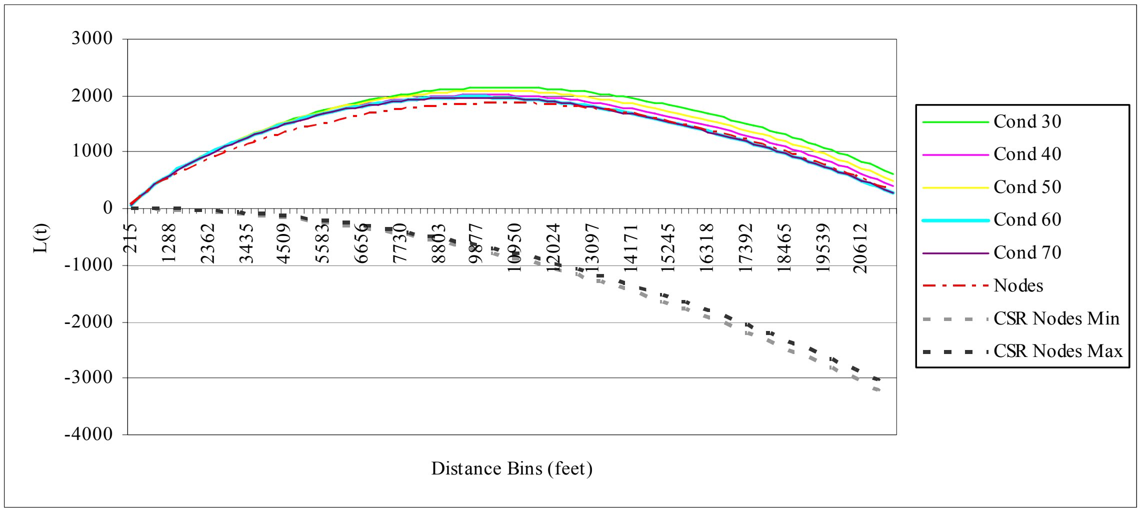
Ripley's *K* offers a more formal test of the form of the distribution across versions and experimental conditions.[44] Figure 4-2 compares the concentration of street robberies under all five of the experimental conditions to the concentration of the street network's nodes, and to a reference distribution describing the amount of concentration that would be expected under CSR.[45] As in the original study, street nodes are significantly more concentrated then would be expected under CSR (Groff, Forthcoming-a). Results of the Ripley's *K* function indicate that there is a high degree of concentration in street robbery locations across all five conditions. The street robbery distribution lines for the Simple (Figure 4-2a) and Temporal (Figure 4-2b) versions of the model are very similar to the one for street nodes in general and to each other.

[44] The reported Ripley's *K* functions are generated using CrimeStat III (Levine, 2005). Following the original study, no edge correction is applied since approximately three quarters of the perimeter of Seattle is bounded by water.

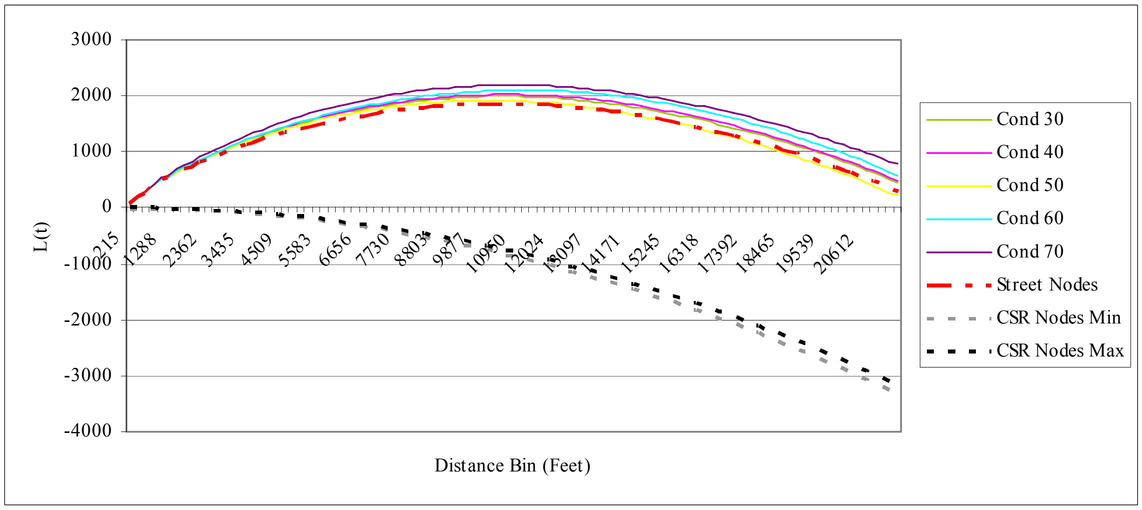
[45] The CSR *K* function distribution is generated by using a uniform random number generator to create 100 distributions with the same N as the observed distribution, in this case N=16,035 (Levine, 2005). A significance level of $p < .05$ is used. The random distribution generated under CSR is truly random in that any location can be selected, not just an intersection.

Figure 4-2: Ripley's *K*: Distribution of Robbery Events

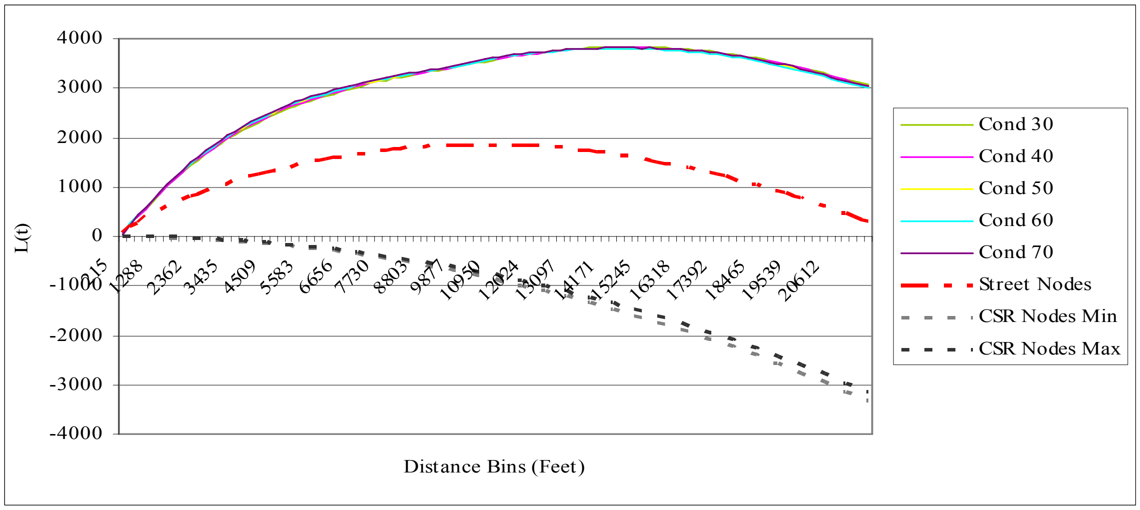
(a) Simple



(b) Temporal



(c) Activity Space



In the Simple version of the model, robberies are most concentrated when society spends 30% of time away from home and the concentration decreases as time spent away from home increases while the 70% condition is the most clustered for the Temporal version. Temporal version robberies track the clustering in street nodes until about a quarter of mile when they become and then remain more clustered at all distances. The one exception is for condition 50 which exhibits the same level of clustering as the street nodes at distances less than about one mile and greater than two miles. The Activity Space *K* function lines are significantly more clustered than the street nodes at all distances and under all five experimental conditions (Figure 4-4c). Unlike the Temporal and Simple versions, there is very little variation among the individual experimental condition lines for the Activity Space version.

An additional analysis of the distribution of visits (i.e. number of times a civilian agent is at a node) to separate the clustering in street robbery from clustering due to everyday travel patterns shows that the patterns for visits and robberies are very similar across all model versions with robberies exhibiting slightly more clustering than would be expected based on the network. This provides evidence of the existence of additional factors, beyond routine travel, that are contributing to the greater concentration of street robbery events.[46]

The final hypothesis is that *the temporal and spatio-temporal schedules of civilians while away from home change the spatial pattern of robbery events.* It explores the impact of systematically adding temporally and spatially explicit components to agent behavior on the spatial distribution of street robberies. Kernel

[46] Due to space constraints the kernel density and Ripley's *K* results for the analysis of visits are not included in the paper but are available upon request from the author.

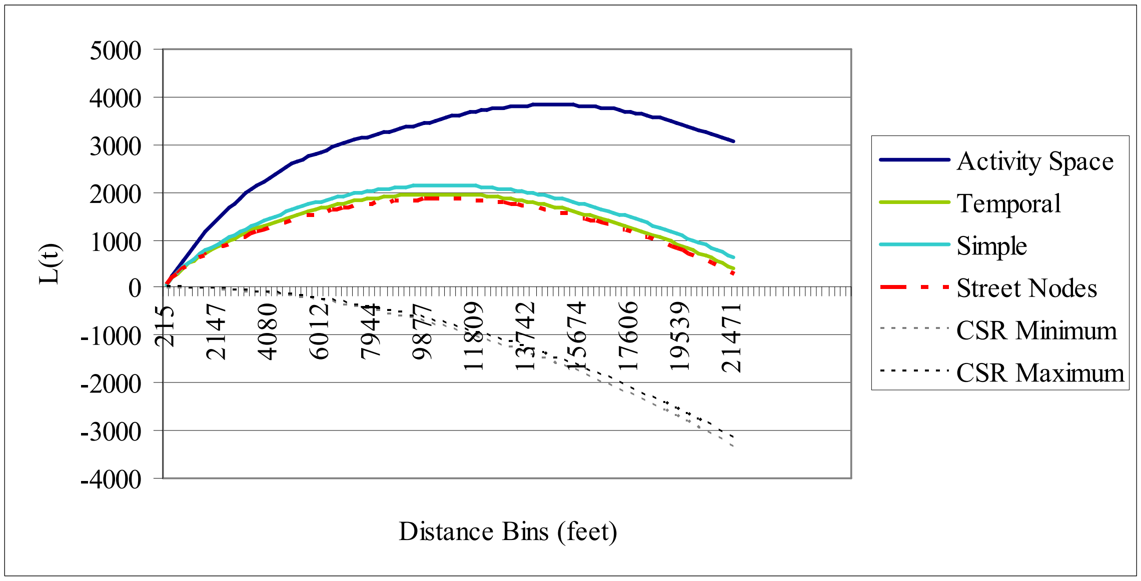
density maps (Maps 4-1 and 4-2) reveal the existence of intra-version and inter-version differences in the spatial patterns of street robbery at the 30 and 70 percent conditions. In general, the Simple condition has fewer clusters then the Temporal version but the clusters represent higher density areas, regardless of the experimental condition. The location of the densest clusters are in the same general area (e.g. in and near the downtown) for both versions but the distribution for the rest of the city is very different. The Simple model has more clusters in the southern part of the city and the Temporal Activity Space has more in northern Seattle. As the only spatially-defined version, the Activity Space version has a pattern distinctly different from the other versions; one that reflects the unchanging activity spaces of the agents in the model.

Another way to examine the differences in the spatial pattern of street robberies produced by the addition of time and then time-space to the civilian's activities is to use the Ripley's *K* calculated earlier but compare the distributions produced by the model versions at the 30 and the 70 percent experimental conditions (Figure 4-3).[47] The street robbery pattern produced by the Activity Space version is significantly more clustered than the other versions, and than would be expected based on the street node network regardless of experimental condition. This result is reasonable, although not predicted; because the Activity Space version specifically restricts civilians to pre-defined activity spaces for the duration of the model run.

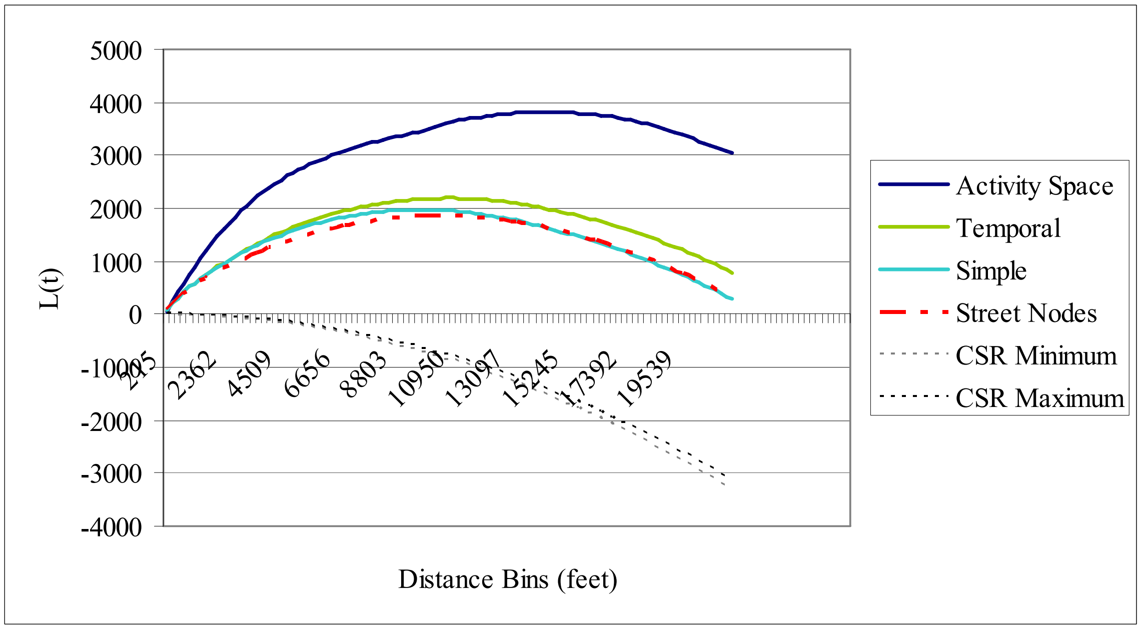
[47] Only the 30 and 70 percent graphs are included in the paper. Results from the 40, 50, and 60 percent conditions are available from the author.

Figure 4-3: Ripley's *K* Analysis: Distribution of Street Robberies by Model Version

(a) Condition 30: Society Spends 30 Percent of Time Away From Home



(b) Condition 70: Society Spends 70 Percent of Time Away From Home



On the other hand, the pattern of clustering in the Simple and Temporal versions relative to one another changes depending on time spent away from home. Although both are more clustered then the street nodes at distances under two miles, the strength of that clustering varies with experimental condition. The Simple version exhibits higher levels of clustering at distances under two miles for 30 percent condition. At 70 percent the results are identical at shorter distances but after about a mile they two versions switch roles and the Temporal version exhibits greater clustering because of the smaller activity spaces. Overall, temporal constraints reduce clustering in the distribution at distances between one-half and two miles but only in societies in which civilians spend up to half of their time away from home. In sum, adding spatial constraints produces a much larger effect than adding temporal ones.

## 6.6 Sensitivity Test Results

Similar to results for the original model, the new model versions are robust to changes in parameters and random number seeds (Groff, Forthcoming-a). The absolute number of robberies increased or decreased depending on the parameter being varied. However, findings related to RAT's core proposition remain consistent across all the tests except two, lending additional support for robustness of the model even as parameters are varied. First, one of tests varying the random number seed did produce a significant result for the Activity Space version. Otherwise, the results of the model are shown to be robust to changes in the random number seed.[48] Second, increasing the time an agent with criminal propensity has to wait to commit another street robbery made the ANOVA for the Temporal version non significant and pointed to the importance of timing in the decision to offend. These were the only changes from manipulating the base model parameters.

[48] The numeric results are available upon request from the author.

## 6.7 Explanations for the Emergent Patterns

Explanations for the findings just described have their roots in the simple rules governing agent behavior and interaction. The separate influences of temporal and spatio-temporal schedules and the movement rules of the agents underlie the finding that time and space each have an effect on the incidence and spatial distribution of street robbery events stems. Temporal schedules reduce the time agents are ‘at risk’. Agents with a temporal schedule spend time at their activity nodes which means they are not vulnerable to street robbery for as much time and they travel shorter distances each day. These changes to their behavior directly impact the number of times they end up at a place where a crime might occur (i.e. the number of convergences) and thus, the rate of street robbery in society. They also change the spatial distribution of robbery events by making the potential path space of the agents in the Temporal version smaller.

Empirically-informed spatio-temporal activity spaces in the Activity Space version of the model increase convergences in two ways. First, they increase the clustering of agent’s homes, jobs, and activity nodes as compared to a random distribution. As a result, civilians are funneled along many of the same roads to reach many of the same areas. Second, the use of a defined activity space embodies the routine nature of daily travel and its restriction to a potential path area in which individuals can travel to all the required locations within their time schedule (H. J. Miller, 1991). Civilians following these predefined paths are by definition then converging only with other civilians’ paths that physically intersect their own. In this way, spatially-defined activity spaces act to: 1) concentrate the activities of agents sharing the same activity space; 2) increase the frequency of convergence of agents in the model; and 3) increase the deterrence effect of police who are in those high concentration areas.

Adding a spatially-defined activity space produced results that did not support RAT's premise that as time spent away from home increases, crime will increase. The explanation for this finding is probably in the implementation of the model rather then the theory itself. Currently, the activity spaces of civilians are unrealistically consistent which concentrates their interaction to one daily path. While this type of activity space may be accurate for some small proportion of individuals, accessibility research indicates there is typically more variety in daily paths (Kwan, 1998; Weber & Kwan, 2002). As a consequence of this concentrated interaction in the model, the same civilians with and without criminal propensity meet again and again. As the civilians with criminal propensity accumulate more wealth than the non-criminal civilians, especially in situations where there are only two agents (criminal and non-criminal civilian), no crime will occur. Thus while the frequency of convergence in the model continues to increase dramatically, the rise in street robberies is much slower and there are not significant differences as society spends more time away from home. Adding to the variety of agent paths or increasing the number of agents in the population might produce a finding consistent with routine activity theory but must wait for future versions.

## *7.0 Discussion and Conclusion*

This research extends earlier work that presented the case and provided the method for testing routine activity theory using simulation and found support for the theory’s main premise that the shift of routine activities away from home increases rates of street robbery (Groff, Forthcoming-a). The approach used was unique in that it incorporated all travel behavior and interactions of individuals when usually only the situations in which a crime occurs are studied. However, the earlier study left unexamined the temporal and spatio-temporal aspects of routine activities in the model.

This paper extends the earlier work by creating two additional versions of the basic model of street robbery based on an existing methodology for ‘situating simulation’ (Groff, Forthcoming-b). One version adds time schedules to the civilians that more realistically reflect the actual vulnerability to the crime of street robbery by restricting their ‘at risk’ status to times they are traveling or participating in an activity that requires them to be outdoors (e.g. walking in the park, shopping along the street, etc). The other version defines both the spatial and temporal schedules of the civilians by having them travel among a set of locations spending the same amounts of time traveling and at activities as in the Temporal version. In this way, the current work is able to: 1) test the core premise of routine activity theory that shifting routine activities away from home increases street robbery; 2) explore the impact of progressively more complex temporally and spatio-temporally explicit activity spaces on the incidence of street robbery; and 3) examine the influence of the first two on the spatial distribution of street robbery events.

The resulting analyses provide strong support for the important role of time and particularly space-time in the definition of activity spaces. Specifically, routine activity spaces that include time or time-space produce different quantities of street robberies with dissimilar spatial distributions across Seattle. In other words, as expected based on theory, the introduction of temporal and spatio-temporal elements to the activity spaces of the civilians in the model causes changes in both the frequency and spatial distribution of street robbery events. However, the simulation provides inconsistent evidence for the plausibility of routine activity theory; while the results of the Temporal version support the plausibility of RAT's main premise, those from the Activity Space version do not.

Further examination of mechanisms underlying the observed outcomes are key to achieving a better understanding and deserve further examination, particularly in terms of model enhancements that would more completely reflect theory about human behavior in space-time. The following discussion highlights the directions for future work with the greatest promise for better representing activity spaces in the model.

Of the many ways in which complexity could be added to the model, incorporating a more realistic representation of time is perhaps the most intriguing. Adding time of day would allow for a more fully developed representation of temporal constraints to be included in the model (H. J. Miller, 2005). Even the naïve version of time implemented here this research demonstrates that temporal constraints influence both the rate and spatial pattern of street robberies. Other research provides the basis for more sophisticated representations of temporal constraints on offender behavior by illustrating how they are important in shaping the spatio-temporal patterns of opportunity-based crimes (Ratcliffe, in press). The requirement that an individual arrive at work at a certain time, the restriction of a lunch hour, the need to pick up dry cleaning, and to stop at the grocery store before arriving home to cook dinner; all these tasks constrain the spatio-temporal activities of individuals. Individuals also have discretionary time (i.e. time not allocated to a task or to travel) that is important to include in any model. Incorporating time of day would have the added benefit of enabling the model to reflect more general temporal patterns such as the fact that more people are away from home during the day and early evening than during the night time (i.e. when most people are sleeping).

Additional temporal constraints related to mode of transportation and characteristics of the street network could enhance the notion of 'at risk' status. Individuals who are walking or using public transportation are at greater risk of street robbery than those in automobiles. Mode of transportation also impacts the distance that can be traveled in a particular time period which could be included in the model. The inclusion of speed limits and one-way streets would better provide greater realism in creating the routes among locations.

Beyond temporal constraints, the representation of activity spaces in general deserves more attention. The current implementation does not reflect the multi-layered complexity that represents the human experience and decision-making. For example, Hägerstrand's more fully descriptive notion of human behavior in which people have intentions that they are trying to realize through their behavior and these future intentions influence present behavior is missing (Hägerstrand, 1975). Nor does the model include rational choice perspective's recognition of the role that 'familiarity' plays in the decision to commit a crime. These enhancements require the use of genetic algorithms which enable the agents to have the knowledge of their past and the experience that comes with such knowledge.

Ideally, activity spaces would be dynamically generated during each day of the model. This would allow the activity spaces to emerge during the course of a simulation and become the object of a study. Unfortunately, this strategy would require that the necessary software classes be developed so the appropriate data structures could be accessed during model runs. In the interim, there are several enhancements that can be implemented such as including more agents and generating multiple activity spaces per agent. Including more civilian agents would increase the potential for convergence and better reflect the density of the city. By increasing the number of activity spaces available to agents, the model will better reflect the variety found in activity spaces. For example, activity spaces are usually different during the week then on the weekend.

Finally, replication is necessary to improve the external validity of the model. The testing of different cities with varying street networks and land use patterns would provide additional insight into the role of the built environment. As it is, this research develops two new versions of a basic model of street robbery demonstrating the important role of space, time, and the built environment in structuring activity spaces which in turn mediate convergences and the crime patterns that result.

More generally, this research demonstrates how geographical perspectives can inform the spatio-temporal representation of human behavior in a criminological theory and in doing so advance the body of knowledge in both disciplines. The research breaks new ground by extending an existing model so that it is capable of accommodating both the empirically-based activity spaces and the individual-level interactions necessary to represent crime events. The resulting analyses provide strong support for the importance of considering time and space when modeling human behavior. Indeed, progressively adding first temporal and then spatio-temporal activity spaces significantly changes both the incidence of street robbery and the spatial distribution of the events. These findings provide evidence for the importance of including the individual-level, spatio-temporal aspects of human spatial behavior in crime event models.

# Chapter 5: Discussion and Conclusions

This final chapter discusses and presents conclusions regarding the major findings from the program of work undertaken here. First, the motivation for the research is briefly explained. Next, the research questions are examined and the major findings are discussed. The potential limitations of the study are presented in terms of their impacts on the findings. Theoretical implications of the study are described. The chapter ends with a discussion of future research plans including approaches to assessing the results of these types of models.

## *5.0 Background*

The characteristics of human behavior strongly influence where, and with whom, individuals converge in space and time. A major theory in criminology, routine activity theory (RAT), recognizes the role of *routine activities* in bringing together the other elements necessary for a crime (i.e. *suitable target, likely offender* and *capable guardian*) (Cohen & Felson, 1979). However, the ability of researchers to empirically test a micro-level theory, such as RAT, has been hampered by the lack of individual-level data (J. E. Eck, 1995a). Previous tests were also handicapped by the need for techniques that can accommodate heterogeneous individuals, non-linear processes and dynamic interaction occurring in space-time.

Simulation modeling provides a framework for addressing these challenges. Since a simulation involves an artificial society, it negates the need for individual-level data about activities. Recent developments have integrated ABM and GIS tools enabling researchers to represent individual-level behavior within its environmental context. This research examines the core propositions of RAT through the creation of an ABM/GIS model of street robbery in Seattle, Washington. In doing so, the study demonstrates how theoretical assumptions can be formalized in an agent-based model and tested via systematic manipulation of those assumptions to discover if the theoretically-predicted outcomes match the model outcomes.

## *5.1 Major Questions and Findings*

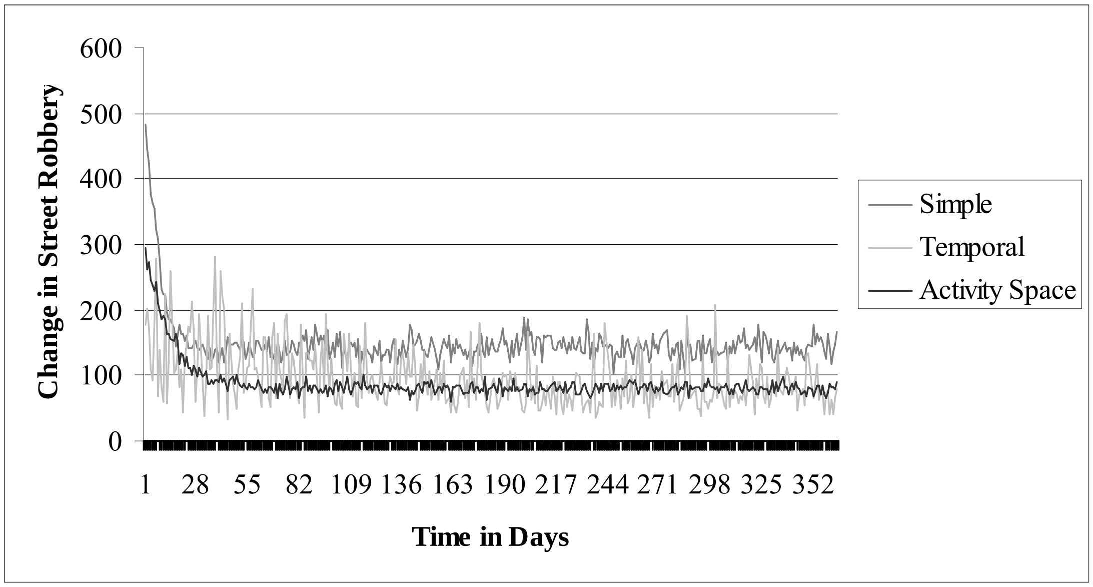
Three research questions are examined in the study. The first investigates whether the shift in routine activities away from home increases street robbery. The second asks whether the spatial distribution of street robberies changes as people spend more time away from home. The third, explores how the spatio-temporal structure of routine activities influences the incidence and spatial pattern of street robbery.

Three major findings emerge from these efforts. First, support for routine activity theory's core proposition depends on the type of schedule constraints placed on the agents. When agents have no constraints on their travel or when they have only temporal constraints (i.e. the Simple and Temporal versions), the number of street robberies increases as the agents spend more time away from home. However, when the agents are assigned spatio-temporally defined activity spaces, the incidence of street robbery still increases but the differences among the experimental conditions are not statistically significant. Therefore, the findings provide support for routine activity theory's core proposition but not when the agent's activity spaces are spatially constrained. This finding also provides evidence of the importance of the spatial component of routine activity in structuring where and with whom convergences occur.

The finding of non significance for the Activity Space version has theoretical implications. It demonstrates that spatial constraints counteract the influence of increasing time spent away from home. Not completely, since crime continues to increase with time spent away from home in the Activity Space version. However, enough to reduce the differences between the experimental conditions and render them non significant. Thus it is the spatio-temporal etiology of routine activity, and not just the gross amount of time spent away from home, that underpins macro level robbery rates.

The implementation of activity spaces in the model is one potential source of explanation for the lack of significant findings for the Activity Space version. The maximum of two only potential activity spaces (i.e. when employed and when unemployed) constrains the spatial extent of agent travel to an unrealistic degree. Consequently, during any model run specific agents can only converge with the relatively few other agents whose activity spaces intersect their own. While activity spaces are somewhat static, it is the degree to which activity spaces are constrained that is the issue. In the model, the repeated interaction of the same agents quickly causes the offender agents to gain more wealth than the civilian agents, so that no crime occurs when only two civilians converge and the offender has more wealth. As a result, increasing convergences do not translate into higher numbers of robbery. While this phenomenon is present in all three versions of the model, it is most pronounced in the Activity Space version. Figure 5-1 shows the precipitous drop in daily robbery statistics for all three versions. Three potential strategies that may ameliorate this phenomenon are to: 1) make the wealth distribution for citizens reflect criminal propensity by assigning offenders lower wealth; 2) increase the number of civilian agent activity spaces available for agents; and 3) boost the number of civilian agents in the model.

Figure 5-1: Daily Change in Street Robbery Events



The second finding is that the spatial and temporal structures of routine activities have separate and unequal impacts on the convergence of the elements necessary for a crime. Consistent with the first finding related to incidence of street robbery, the spatial distribution also changes as time away from home increases but only for the Simple and Temporal versions of the model (i.e. when there are no spatial constraints). The maps of kernel density show changes in the locations of high density areas as the time away from home increased and the Ripley's *K* results indicate changes in the clustering of street robbery across experimental conditions.

However, additional time spent away from home in the Activity Spaces version produces only small changes in pattern but large increases in the intensity of clustering. Thus, spatially constrained activities that reflect opportunity structures in a community are the source of generally stable hotspots that increase in intensity as time spent away from home increases.

Thirdly, temporal and spatio-temporal constraints have a differential influence on the incidence of street robbery. As compared to the Simple version, in which agents are either at home or not at home, the addition of temporal schedules for civilian agents reduces the incidence of street robbery by about 77% and changes the distribution of street robbery events. This result provides evidence in support of Ratcliffe's (in press) hypothesis that temporal constraints are a major source of observed patterns of opportunity-based crime. When spatially defined activity spaces are added to the model and the temporal schedule for each agent is held constant, the separate and even larger impact of space is clearly demonstrated. Spatio-temporally constrained schedules significantly increase the incidence of street robbery as compared to agents with a temporal schedule only and radically change the distribution of street robbery events. The clustering in the spatial distribution of robberies is higher than the other versions and more linear in nature due to concentration along the major travel routes among homes, jobs and activities.

These findings regarding differences by type of schedule constraints are consistent across all five experimental conditions. In other words, regardless of time spent away from home, the type of schedule (i.e. simple, temporal or spatio-temporal) produces significantly different numbers and patterns of street robbery. Thus, the impact of temporal and spatio-temporal constraints on activity is robust with regard to time spent away from home.

There are several potential explanations for these findings. The changes in incidence and pattern could be related to the amount of time the agents are ‘at risk’. The addition of a temporal schedule reduces both the time that civilian agents are ‘at risk’ of being victimized and the time that civilians with criminal propensity have to offend. In this way, temporal schedules constrain the activities of both offenders and non-offenders and directly influence the number and pattern of convergences. Differences in time ‘at risk’ do not explain the increase in street robberies between the Temporal and Activity Space versions because the temporal schedule is held constant between the two.

The explanation for this finding lies in the clustered nature of human activity that is reflected in the Activity Space version of the model. The homes of civilian agents are concentrated in certain areas, they travel to jobs that are clustered in other areas and they participate in activities that have yet another, but still clustered, distribution. The road network acts to amplify this result in that agents traveling to the same area tend to be routed along the same major roads. In this way, the implementation of spatio-temporal routine activity spaces following time geographic principles acts to increase overlap in activity spaces which in turn, increases the frequency of convergence. One interesting side effect of this increased concentration is that police agents who are randomly assigned to patrol in those high concentration areas are able to deter more crimes than when civilian agents are only temporally constrained but randomly distributed (as in the Temporal version of the model). This finding has implications for achieving a better understanding the relationship between police patrol strategies and crime.

## *5.2 Significance of the Research*

This examination of how the spatio-temporal nature of human activity influences the incidence and distribution of street robbery events breaks new ground on several fronts both theoretical and methodological. Theoretically, the research provides discoveries stemming from the ability to test both the micro and macro aspects of routine activity theory simultaneously and to explicitly examine the role of temporal and spatial constraints. First, the model runs confirm that if the number of offenders and their motivation is held constant, as people spend more time away from home, the number of street robberies will increase. The differences among the experimental conditions are significant in both the Simple and Temporal versions, but not for the Activity Space version. In other words, when the core propositions of routine activity theory are implemented in an artificial society, there is mixed support for its core proposition and that support hinges on the spatio-temporal aspects of human behavior. Thus the second discovery is that both temporal and spatial constraints play a key role in determining the incidence of street robbery and should be included in future empirical studies that aim to tease out the role of routine human behavior in robbery events.

Third, the findings from the research demonstrate the importance of the street network and the distribution of opportunities in structuring travel and street robbery. When the activity spaces of the agents reflect the distribution of opportunities, the clustering of the spatial distribution increases dramatically confirming the important role of opportunities in shaping spatial patterns of street robbery. These findings lend credence to the inclusion of ‘place’ in later formulations of routine activity theory (Felson, 2001;2002).

Methodological advances include the following. First, the research provides a well-documented example of how a simulation model offers a unique opportunity to formalize theories from multiple disciplines and then compare the theorized outcomes to the model outcomes. Specifically, the resulting effort creates a formal representation of routine activity theory that can be tested and enhanced in an artificial environment before the expense of empirical data analysis is undertaken. The agent activity spaces developed for this research provide the foundation subsequent, richer representations of activity spaces. In addition, this work facilitates future research by providing written and copious documentation of the model assumptions; a resource that is necessary for replication and testing. In sum, this effort fills the need identified in previous research for the publication of example models with documentation (Axelrod, Forthcoming).

Second, the research demonstrates the value of ‘situating’ simulation. A methodology is presented to develop a representative society that interacts in a real cityscape. In this case, Seattle, Washington provides the environment in which the agents live their lives. Activity spaces are created based on the distribution of population, housing, recreation, retail and services in the city; thus, effectively incorporating the influence of land use on the activities of agents. In addition, agents can move along the vector GIS streets of Seattle. This enables researchers to use their vector geographic data sources directly without having to convert to a grid-based system. Together these advancements make it possible to compare the model outcome under three different assumptions of spatio-temporal activity constraints. Under the first assumption the agents move randomly, under another they move randomly but have a temporal schedule and under a third they have both spatial and temporal constraints on their activities. In this way, the research design enables the impact of time to be tested separately from the impact of space.

Third, the computational laboratory framework enables the first test of routine activity theory based on individual-level data. Although conducted using representational agents rather than empirical data, the computational laboratory framework permits a high level of scientific rigor to be applied to design and testing of the model and the analyses of results. The result is a simulated laboratory environment in which the emphasis is on increasing our understanding of the processes behind observed patterns. Different aspects of the model can be changed while all others are held constant in order to isolate, as much as possible, the impact of that one variable from all others. In general, the method used in this research represents an interim testing ground between the verbal formulation of the theory and the testing of theory with empirical data. While these tests do not result in a determination of whether a theory is empirically valid, they do provide a way to strengthen the theory prior to empirical testing. A process that has been described in the literature as 'elaborating' (J. Eck, 2005) or 'experimenting' on theories (Dowling, 1999).

## 5.3 Assessing the Model Results

As stated earlier the goal of this research was to operationalize theoretical assumptions and then test whether the results generated from an artificial society match what routine activity theory would predict. However a related question concerns strategies for assessing the validity of ABM-produced results. One approach is to compare the distributional properties of the rates and patterns of street robbery to empirical data. This approach depends on being able to obtain reliable empirical data. In this case, the required data would be official crime data on street robbery. The reliability of official crime data is questionable and varies by type of crime (Gove et al., 1985; Kerlinger & Lee, 2000). Violent crimes such as robbery are more likely to be reported to police by the victim and to end up as an official crime report. However, victims who are engaged in illegal behavior themselves are less likely to report being robbed. The shortcomings of official crime data lend credence to assessing validity by comparing the characteristics of the distributions produced by the model with ones produced from empirical data.

Both spatial and aspatial characteristics of distributions can be used to compare model and empirical results. A variety of studies have found that the spatial distribution of crime is clustered across space (P. L. Brantingham & Brantingham, 1999; Sherman, Gartin, & Buerger, 1989; D. Weisburd & Green, 1994; D. L. Weisburd, Bushway, Lum, & Yang, 2004). Most frequently, a relatively few places are responsible for a large proportion of the crime. In addition, these studies have demonstrated the existence of hotspots (i.e. clusters of crime locations) that often persist over time. In line with these well-known characteristics of the spatial distribution of crime, the model results produced by this research show evidence of hotspots and the concentration of robberies at a relatively few locations.

Aspatial characteristics can also be used to compare model results with empirical ones. These comparisons could include characteristics such as the rate of street robbery per population, rate of repeat victimization, measures of offenders as victims etc. While empirical street robbery data were not available for Seattle, one model result matched that of empirical studies. Just as in real life, criminals in the model were victimized at a higher rate than non-criminals (Deadman & MacDonald, 2004).

When model results share characteristics with empirical ones, the credibility of the model increases. Consistent empirical and model-produced findings demonstrate that the simulated mechanisms produce distributions that share characteristics with empirical ones. However, matching distributions is not a sufficient criteria for validation since a different model could also produce comparable patterns (Troitzsch, 2004). In sum, establishing model credibility is an incremental process that involves multiple comparisons and is not an exact science.

## *5.4 Possible Limitations of the Research*

There are several potential limitations to the research, some stem from the use of ABM and others are related to the implementation model created and the reliance on one site's data. Related to ABM in general, the meaning of findings from an artificial society inspires debate. Since the goal of this research was to explore the assumptions of routine activity theory, the findings demonstrate to what extent the theory is plausible. This study provides evidence in support of routine activity theory but additional empirical testing is required to determine the empirical validity of the theory.

Another characteristic of agent-based models is that their findings are constrained by the assumptions and rules on which the model is based. This study relied on empirically-based parameter values whenever possible to strengthen their validity. In addition, the study used sensitivity tests to characterize the impact of parameter value changes on the model outcomes, and to determine whether the findings from a single run are representative (Axelrod, Forthcoming; Gilbert & Troitzsch, 1999). To test the robustness of the research finding five different parameters were systematically varied. Only one, time an offender had to wait before committing another robbery, changed the experimental results and then only for the temporal version of the model. Four other input parameter values were tested but none of those changed the findings. Additional sensitivity testing was conducted by manipulating the random number seed four more times. The model findings were robust across random number seed tests with one exception; under one random number seed the Activity Space version became significant.

Agent-based models rely on random numbers and random number distributions to provide a stochastic element to the simulation. Similar to the choice of parameter values, the choice of distribution (e.g. Uniform, Poisson, Normal, etc.) and the moments of the distribution (e.g. mean, standard deviation etc.) have implications for model results. In this study, the wealth distribution demonstrated a large influence over the model's time to equilibrium. The model assumed wealth is distributed normally across all civilians, quickly producing a society in which the offenders have more wealth then the civilians so the offenders are victimized at a higher rate. Subsequent experiments with a two-tiered wealth distribution (one for civilians with criminal propensity and another for those without) under the Simple version of the model find that it lengthens the time to equilibrium and reduces repeat victimization among offenders but further study on this aspect is needed.

Two limitations of the model as implemented are related to the characterization of civilian agent activity spaces. First, software limitations forced the creation of static rather than dynamic activity spaces for the Activity Space version of the model. As discussed in the findings, this severely limits the variety of travel that agents undertake during a model run to only one of two routes and in so doing, concentrates agent activity and limits convergences to the same set of agents. Because of this limitation, the findings regarding Activity Space version should be interpreted with caution.

The random movement in the model has two limitations. First, the agent only considers the adjacent nodes so they only can move one node per minute. Second, there is no prohibition against back-tracking by agents so they can move back to the node they had occupied in the previous minute. Together these implementation decisions may lead to smaller, less realistic activity spaces for agents that are moving randomly. Future implementations should consider giving agents who are moving randomly the same ability to move more than one node as the agents who have directed movement. The implications of these shortcomings are revisited in the suggestions for extensions to the model.

Finally, the generalizability of the findings is limited by the nature of the data and the reliance of a single site. The single site analysis was appropriate for an initial effort such as this one but that decision restricted the ability to make statements about the plausibility of model assumptions in cities with different street networks and distributions of jobs, homes, and activities. In addition, the base data for the distribution of jobs were for larger geographic units (i.e. zip codes) than those for homes and activities (i.e. blockgroups). The validity of the assumption of homogeneous distribution of data across each unit becomes weaker as units grow larger which in turn, decreases the potential for the allocation of job locations to agents to reflect the actual distribution of jobs within each zip code. This is a relatively minor limitation since the goal is to distribute job locations of agents to areas proportionately, not exactly.

## *5.5 Next Steps*

As far as enhancements of the current model, a few of the directions highlighted by the findings of the model are discussed first and then a couple of other promising directions are noted. The research endeavor reveals three ways in which the model could be enhanced. One is to provide civilian agents with a greater variety of activity spaces, which could be done by simply creating a more activity spaces for use in the model. A second is to increase, as much as possible given limitations of computing power, the number of civilian agents in the model to better reflect the potential for convergence and bring the civilian/police ratio closer to what is found in Seattle. A third would assign agent’s wealth based on a two-tiered distribution with its roots in the mean income distribution of Seattle. Civilian agents with criminal propensity would get a lower wealth distribution than agents without criminal propensity. In addition, pay would be in proportion to wealth for agents who are employed. These changes would better reflect the existing wealth distribution and how wealth figures into the decision to commit a street robbery. After all, reports of street robbers who commit robbery to make their mortgage payment are rare. Preliminary tests conducted with different wealth distributions demonstrate that a two-tiered wealth distribution reduces the time to equilibrium in the model.

Including situational aspects of the crime event probably represents the most intriguing direction for enhancements of the model. The theoretical basis for including situational characteristics can be found in later formulations of routine activity theory (J. E. Eck, 1995a; Felson, 1986a;1986b;1987) and under rubric of environmental criminology (Paul Brantingham & Brantingham, 1991 [1981]). More recent additions to routine activity theory include a refined concept of guardianship that relies on two additional elements intimate handlers (Felson, 1986a) and place managers (J. E. Eck, 1995b) as well as a recognition of the importance of place characteristics and the distribution of population (Felson, 1986b;1987) in the patterning of crime. Empirical research within these theoretical frameworks has generated a large body of knowledge about the characteristics of places and situations that are related to crime.

The final example of future work discussed here (but only one of many potential ones) is to give the police realistic patrol strategies and measure the relative effectiveness of each. For example, under directed patrol, the police agents would have particular areas to which they are assigned to patrol. Police could not go outside these areas. The outcome crime patterns using this strategy could be contrasted with a *place-based* or *hot spots* policing strategy (Braga, 2001; Sherman et al., 1989; Sherman & Weisburd, 1995; D. Weisburd, Maher, & Sherman, 1992). These police agents would be assigned to small areas of Seattle at which the concentration of street robbery is the highest (based on model output). Changes in the incidence street robbery could be observed. In addition, the geographic phenomenon of displacement and diffusion could be investigated in depth. These topics are of great interest to both geographers and criminologists.

In sum, this investigation creates as many questions as it answers. Interesting findings that were not directly related to the hypotheses must wait for elaboration in later analyses. For instance, some agents with criminal propensity never commit a street robbery while others commit hundreds. What are the differences between these agents? In the same vein, not all civilians are victims of street robbery yet others are repeatedly victimized. Further, many of those repeat victims are agents with criminal propensity, a finding that is in line with ethnographic studies that indicate criminals are very often victims depending on the situation.

To end, the current effort introduces a new way of strengthening theories through testing in an artificial society prior on empirical tests. This method shows great promise for investigating a multitude of interdisciplinary research questions that cut across the social sciences.

# Appendices

*Appendix 1: Street Robbery Model Documentation: Simple Version*

**Street Robbery Model Documentation: Simple Version**

This documentation explains the Street Robbery Simple Model from a programming perspective following guidelines for sharing simulation research in Axelrod (Forthcoming). A general description of the model is provided in the main text. The theoretical basis for the model is provided in Groff (Forthcoming-a). Details on the implementation of agent movement on a vector network are in Groff (Forthcoming-b). The model is created in a flavor of RepastPy called Agent Analyst.

## 1. REPAST PY AND AGENT ANALYST

RePast Py was developed by Argonne National Laboratories to provide a Python-based syntax for rapid model development. The software uses a hybrid language dubbed Not Quite Python (NQPY). The language uses Python-syntax to access Java classes. Agent Analyst is an extension to RePastPy that can be used as a toolbox in ESRI's ArcGIS software. Agent Analyst provides the ability to use data from a geographic information system (GIS) in agent-based models. Public releases of final versions of software are available at http://repast.sourceforge.net/download.html. For general information on RePast please see http://repast.sourceforge.net/index.html. The Street Robbery Model presented here was developed with a beta version of Agent Analyst which utilized Java 1.2.4_06, Python 2.3 and ArcGIS 9.1. The documentation assumes some familiarity with these languages and software products.

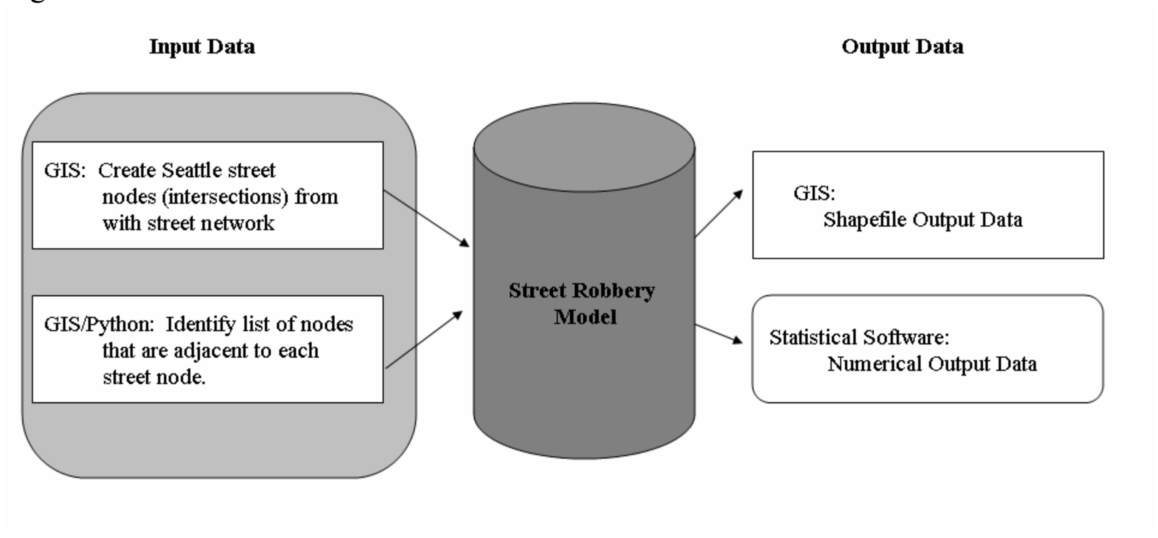
## 2. OVERVIEW OF THE MODEL

The Street Robbery Simple model is based on routine activity theory (RAT) (Cohen & Felson, 1979) and creates a simulation of how an individual agent's decisions on whether or not to commit a street robbery translate into macro-level crime patterns. RAT identifies four elements necessary for a crime to occur. Specifically, the *routine activities* of individuals determine which individuals are at the same place, at the same time. For a crime to occur there has to be a *motivated offender*, a *suitable target* and the lack of *capable guardians*. Each of these elements is represented in the model. A series of experiments are conducted to test whether the outcomes from the model match what the theory predicts (i.e. whether crime will increase as people spend more time away from home). This prediction is tested by systematically changing the amount of time that the society of agents spends away from home. For example, in the first of five conditions agents spend 30% of their time away from home. The results from this model are compared to societies in which agents spend 40%, 50%, 60%, and 70% of their time away from home.

There are two types of people in the model, civilians/citizens and police/cops. [49] Civilians have two characteristics, wealth and criminal propensity. Two-hundred of the one thousand civilians in the model are assigned criminal propensity; they evaluate each situation for the potential to commit a street robbery considering the level of guardianship (formal and informal) and the suitability of the target. Civilians with criminal propensity can take on the role of offender, target, or informal guardian in any situation; while those without can only play the roles of target or guardian. Police have only one role, formal guardians.

The model uses empirical data to inform the movement of agents. All of the agents in the model travel randomly. Citizen agents start at and remain at home for a set period and then begin traveling randomly for the rest of each day. They begin the next day at the ending point of the previous day. Police agents travel randomly without stopping. Figure 1 provides a graphical view of the general data flow. Data describing the street nodes and the neighbors of each node are added into the Street Robbery model and used to support agent movement on Seattle's street network.

Figure 1: Data Flow


Input Data
Output Data
GIS: Create Seattle street nodes (intersections) from with street network
GIS/Python: Identify list of nodes that are adjacent to each street node.
Street Robbery Model
GIS: Shapefile Output Data
Statistical Software: Numerical Output Data

There is one main model and four classes of agents in the Street Robbery Model. Three of the classes consist of generic agents – cops, citizens and active nodes. One class is made up of vector agents, places. Vector agents have an inherent spatial property that is needed in the simulation, generic agents do not. Each of the classes has a set of actions, a schedule that controls when the actions run, and a set of fields. The actions control a variety of functions necessary to the running of the model including agent initialization, agent movement, and agent decision-making. The actions can be scheduled to run at each model tick, at a specified interval, or just once during the course of a model run. The fields contain the data that are available to

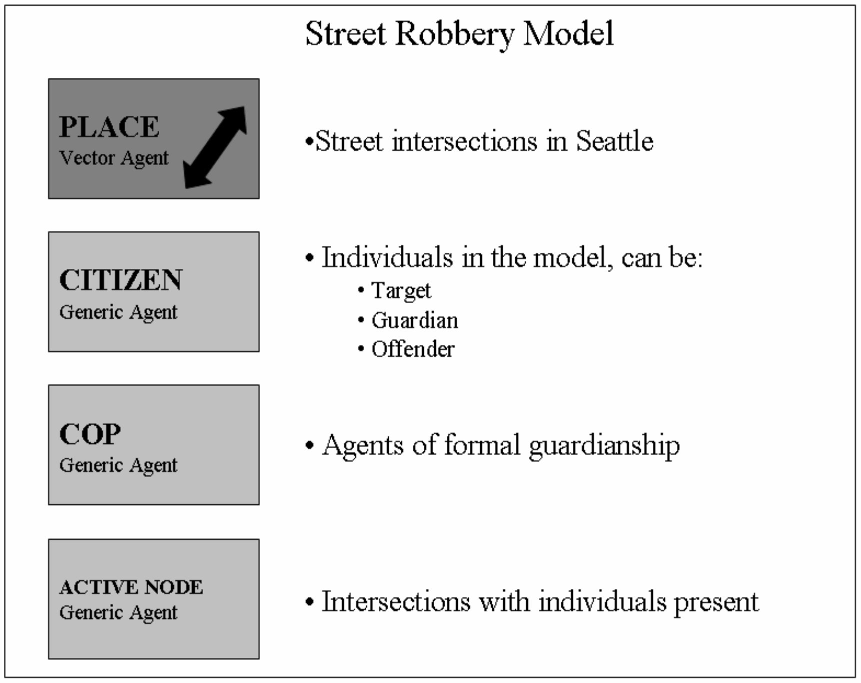
[49] The terms civilian and citizen describe the population of the city who are not police/cops. The general term describing the population was changed to civilian after the program was written and the code has not been changed to reflect that evolution. The same situation is true for the use of the terms police and cop both of which refer to law enforcement officers.

describe the members of each class. There is also a Sequence Graph that is used to graphically display the number of robberies as they occur during the model run.

## 3. MAIN MODEL AND AGENT CLASSES

There is one main model and four classes of agents in each of the versions (Figure 2). Three of the classes consist of generic agents – cops, citizens and active nodes. One class is made up of vector agents, places. Vector agents have an inherent spatial property that is needed in the simulation, generic agents do not. Each of the classes has a set of actions, a schedule that controls when the actions run and a set of fields. The actions control a variety of functions necessary to the running of the model including agent initialization, agent movement and agent decision-making. The actions can be scheduled to run at each model tick, at a specified interval or just once during the course of a model run. The fields contain the data that are available to describe the members of each class.

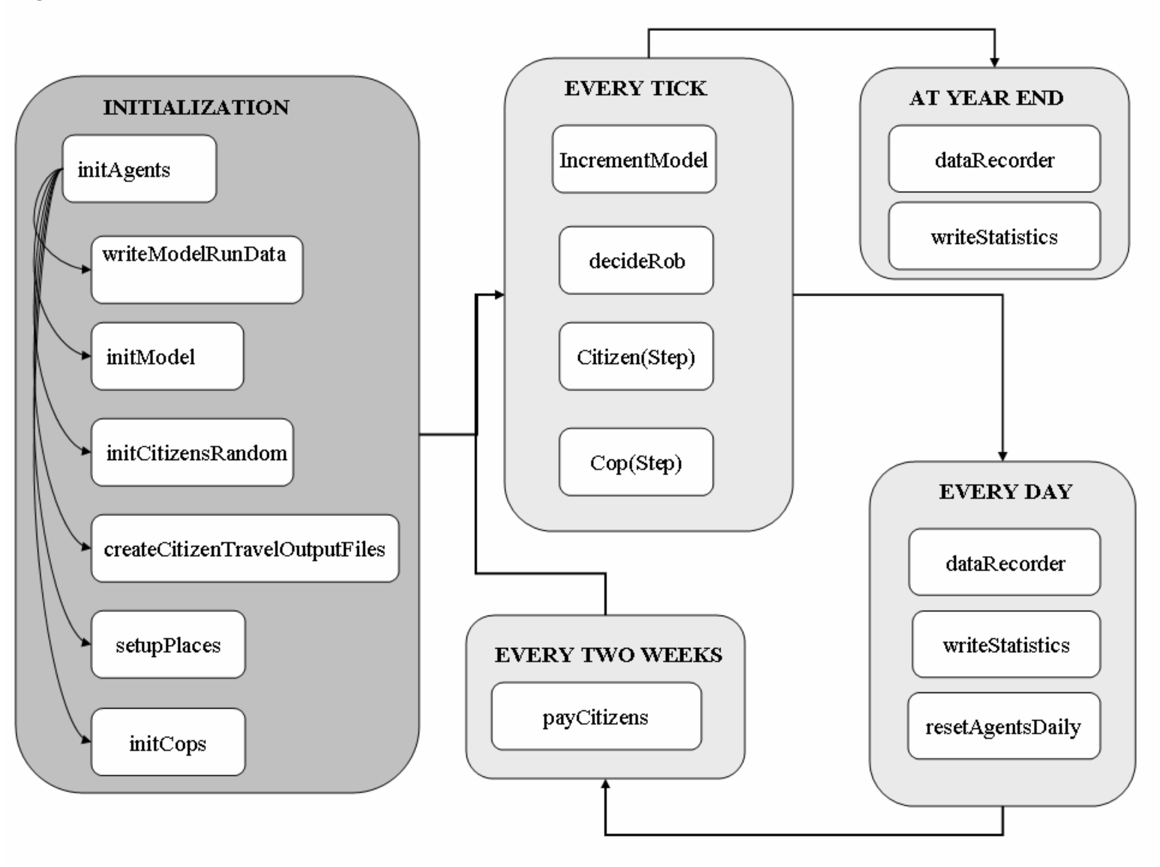
Figure 2: Street Robbery Model Classes


Street Robbery Model
PLACE
Vector Agent
•Street intersections in Seattle
CITIZEN
Generic Agent
• Individuals in the model, can be:
• Target
• Guardian
• Offender
COP
Generic Agent
• Agents of formal guardianship
ACTIVE NODE
Generic Agent
• Intersections with individuals present

### <u>3.1 The Main Model</u>

The main model is called StreetRobSimple. It has a display name of Street Robbery Simple. The main model contains all the actions for the initialization of the model. Each of the actions is listed and their function is explained. Fields that are in all capital letters are static/global variables that can be called from anywhere in the model. The main model is also where all the random number distributions are created so they can be called during the appropriate parts of the model run. Please see Figure 3 for a graphical representation of the order of execution.

Figure 3:  Order of Action Execution



<u>Actions:</u>

initAgents()

Main action that calls other actions to initialize the agents.  This is the first action to run in the model.  It calls the following actions: writeModelRunData(), initModel(), initCitizensRandom(), createCitizenTravelOutputFiles(), setupPlaces() and initCops().  Specifies a random number seed and creates the uniform random number distribution with the specified seed.  The action also creates the street message display function (that is currently not used).

updateDisplay()

Changes the display in ArcMap.  The action is currently scheduled to be called every ten ticks.

writeAgents()

This action writes the agent values from the place class to the shapefile. The action is currently scheduled to be called every ten ticks and runs before the updateDisplay() action.  The symbolization settings are created in the properties of the strnodes2 layer in ArcGIS.  The same shapefile is added into the .mxd file twice; once to symbolize the total robberies at a place, another time to symbolize the total number of visits to a place.

setupPlaces()

The primary function of setupPlaces is to initialize a hashmap and identify the neighbors of each place for random movement. First, a hashmap is created to store a list of all the places with strnode-id as the key. Next, the action reads the nodeNeighbors.csv file and associates the set of neighbors with the correct Place (i.e., it populates the field myNeighbors in the Place class).

showMessage()

The showMessage() action enables the display of custom messages to an output window.

incrementModel()

This action is scheduled to execute at every tick of the model. The action does the following: 1) increments model counter; 2) calls the writeOccupiedNodes() and writeCitizenInfoPaths() actions 3) calls decideRob(); and 4) clears the agent list associated with the ActiveNode class

initModel()

The initModel() is called by initAgents() and sets values for constants and static variables in the model some of which are parameters and can be changed through the RePast GUI at model run time.

decideRob()

Contains the code to evaluate who is at the occupied nodes and then decide whether a crime should occur. First, the action checks to find out which agents are at a streetnode by using the ActiveNodes class. Next, the presence or absence of official guardianship (in the form of a cop) is evaluated. Then the presence of an offender is evaluated (list of agents at the node is shuffled so the same agent is not evaluated first each time and if there are two offenders at a node, each has a random chance of getting to decide to commit a crime). Only nodes with two or more agents and an offender are evaluated as far as the actual decision to commit a crime. Program also takes into account whether the agents at the node are ‘at risk’ (which is set in Citizens.step()). If a crime occurs, the action changes the following field values: victim and offender wealth, number of victimizations and total robberies at the place. This action contains a ton of commented out code that can be used for debugging and understanding the operation of the model. The decideRob() action is called by incrementModel().

writeOccupiedNodes()

This action writes out the distribution of agents across street nodes for diagnostics and data analysis. The action is called by the incrementModel() action at every tick (currently commented out).

However, it could be written out less frequently by scheduling the action instead.

initCops()
Creates cop agents and assigns them to a strnode (number) and a location (Place). Action uses a uniform distribution to select the nodes on which to place the cops at the start of the model.

resetAgentsDaily()
At end of the model day, all agents are reset to be back at home so they are at an activity node (home), not at risk and not moving. Action is in schedule to run at an interval of 1,440 (one model day). Although the fields referring to time at activities are vestigial (e.g. timeMain, timeRec1 etc.) they remain because they are written out in other places and would require significant effort to remove.

createCitizenTravelOutputFiles()
Creates two different types of output files to which citizen data can be written. One creates a unique file for each citizen agent to which output can be written by the writeAgentInfoFiles() action (output/citizenX.csv) (not used). The other type of file is a single file to which data can be written at specified intervals to monitor societal-level citizen characteristics (output/citizenChar.csv) and is very helpful in tracking model output. This file is written to by dataRecorder().

writeCitizenTravInfotoFiles()
Uses the files created by createAgentOutputFiles() and writes out information about the individual agents at different points in the model (not currently called).

writeModelRunData()
This action creates a log file that can be used to capture messages and critical statistics during each model run.

writeStatistics()
This action captures the final field values for citizen agents pertaining to activity spaces and crime in a single file. Writes out the aggregate time spent at home, main, rec1, rec2, travel, and exposed; the assigned time to spend at home, main, rec1, rec2, travel; and the Total number of offenses and victimizations. These same statistics are written to individual agent files by the createCitizenTravelOutputFiles() and writeCitizenTravInfoFiles() actions. The action writes to output/statistics.csv which is written one time at end of simulation and is very helpful for understanding victimization, offending and wealth for each agent.

dataRecorder()
This action takes the place of the data recorder that I could not get to work in the model. It records variables that change for society as a whole during the model run such as: number of unemployed agents, average wealth, robbery rate, total victims, total repeat victims, total offenders, total repeat offenders, percent exposed, percent traveling, and number of active offenders. Writes to output/citizenChar.csv

initCitizensRandom()
Sets criminal propensity and wealth characteristics, and assigns values to the time to stay away from home fields using a random normal number distribution.

writeFinalAgents()
Action writes the final shapefile out to the output folder.

writeCitizenInfoPaths()
This action writes out each node visited by an agent during the course of the simulation to a single file. Using Tracking Analyst, the file can be 'played back' to follow the path of the agent. Additional analysis can also be conducted on the size and shape of the agent's random activity space.

<u>Fields:</u>

messageDisplay - uchicago.src.simbuilder.util.MessageDisplay, displays messages while model is running. Parameter
modelStep – integer, counter that keeps track of model steps, 1 minute steps
MODEL_HOUR – integer, number of steps in an hour, 6 x 60 = 360 steps in an hour. Default value = 60.
MODEL_DAY – integer, number of steps/minutes in a day, 24 x 360 = 8,640 steps = 1,440 minutes in a day.
MODEL_WEEK – integer, number of minutes in a week, 7 x 1,440 = 10,080 minutes in a day; 7 x 8640 = steps in a day.
MODEL_YEAR - integer, number of minutes/steps in a year, 365 x 1,440 = 525,600 minutes in a year; 365*8640 = 3,153,600 steps
SOCIETAL_TIMEAWAY – double, Default value = .70,
totRob – integer - Default value = 0, Cumulative number of robberies in the model run. Parameter
placeMap – java.util.hashmap – hashmap of strnode-ids
AGENTS – integer – Default value 1000, total number of agents in the model. Parameter
totDeter – integer, Default value = 0, total number of robberies deterred by presence of cop for the entire model run. For this to increment there had to have been more than one agent and a criminal agent at the node.
totIntersect – integer, total times there were more than two agents at a node and a criminal. Represents the number of potential crime situations and is a running total for model run.

LOG_FILE – java.lang.string – file name to which data are written
REPEAT – integer, default value is 60-time a criminal has to wait before re-offending. Parameter
MIN_GUARDIANSHIP – integer, contains minimum amount of random error in the perception of guardianship by the criminal agent, default value is -2. Parameter.
MAX_ GUARDIANSHIP – integer, contains maximum amount of random error in the perception of guardianship by the criminal agent, default value is 2. Parameter.
MIN_SUITABILITY - integer, contains minimum amount of random error in the perception of target suitability by the criminal agent, default value is -1. Parameter.
MAX_SUITABILITY - – integer, contains maximum amount of random error in the perception of target suitability by the criminal agent, default value is 1. Parameter.
NUM_PLACES – integer, number of street intersections in the model.
COPS – integer, number of police agents in the model. Parameter
WEALTH_MEAN – integer, mean of the wealth distribution for all agents in the model. Parameter.
WEALTH_SD – integer, standard deviation of the wealth distribution for all the agents in the model. Parameter.
SEED – integer, random number seed. Parameter.

<u>Actions that are in Schedule</u>

Every Tick:
- Citizen: step
- Cop: step
- incrementModel

At Interval:
- Tick 20,160:
  - payCitizens
- Tick 1,440:
  - dataRecorder
  - writeStatistics

At:
- Tick 525.600:
  - dataRecorder
  - writeStatistics

End:
- Data Recorder:  dataRecorder: record
- Data Recorder:  dataRecorder: write

Other actions that I have scheduled in the past:
writeAgents – at interval of 20 ticks
updateDisplay – at interval of 20 ticks
writeCitizenTravInfoFiles – at interval of 40 ticks

resetAgentsDaily - at interval of 1440 ticks
writeFinal – called by incrementModel() at model end

## 3.2 Vector Agents

There is only one vector agent class in the simulation and it consists of the set of places (street intersections) in Seattle. To clarify, each street intersection (also called a street node) is a place in the model. There are 16,035 places in the model and they exist in the shapefile called strnodes2.shp that is a point file. This class contains the geographic information about the distribution of agents and robberies (i.e., where the agents and the robberies are located in Seattle).

***Name: Places***
***Group Name: places***

Actions:
None.

Shapefile fields:
ARC_ - integer, internal arc-id used by ArcGIS
STRCL_ - integer, internal node number used by ArcInfo
STRCL_ID – integer, node number used in the model
citiStart – integer, node at which a citizen agent starts the simulation
copStart – integer, node at which a police agent starts the simulation
crimStart – integer, node at which a civilian agent with criminal propensity starts the simulation
the_geom – com.vividsolutions.jts – geometry of point
totPrevent – integer, total number of potential crime situations in which a cop prevented the crime at a node that would have been committed otherwise
totalRob – integer, total number of robberies at a node
totalVisit – integer, total times any agent visited a node

Class Fields:
strcl_ - integer, street node number
myNeighbors – java.util.ArrayList – list of nodes adjacent

Actions that are in Schedule
None

## 3.3 Generic Agents

Overview of Generic Agents

There are three generic agent classes in the model, citizens, active nodes and cops. These classes are not inherently spatial in nature but through object-oriented programming, the members of the classes are associated with members of the Place class. Each of the generic classes is described in this section.

***Name:*** ***Citizen***
***Group Name:*** ***citizens***

The citizen class contains all the citizen agents in the model. One of the interesting facets of this implementation is that citizen agents with criminal propensity and those without are modeled exactly the same as far as movement, initial wealth, and pay. The only characteristic that differs is the presence of criminal propensity and only those citizens who have criminal propensity evaluate criminal opportunities and are able to make the decision to offend. All citizen agents can be victims of street robbery.

Actions:

step()
: This action changes the atRisk, atActivity and moveStatus variables depending on whether an agent is at home or traveling. Dynamically creates the ActiveNode class with each step. Only agents at ActiveNodes are evaluated during the decideRob() routine.

payCitizens()
: Pay citizens who are employed every two weeks.

Fields:

placeNode – Place, associated with vector group of Places
name – string, name of the agent
home – integer, home node of agent
main – integer, work, school or other significant activity node, this value reflects current employment status (emp or unemp)
rec1 – integer, one activity node, this value reflects current employment status (emp or unemp)
rec2 – integer, another activity node, this value reflects current employment status (emp or unemp)
currentNode – integer, holds the strnode_id of node the agent is occupying
criminalPropensity – Boolean, whether or not an agent thinks about committing a crime, default value=false.
timeHome – integer, minutes spent at home
timeMain – integer, minutes spent at work
timeRec1 – integer, minutes spent at recreation one
timeRec2 – integer, minutes spent at recreation two
timeTraveling – integer, minutes spent traveling among activity nodes (assigned as part of activity space)
totTimeExposed – integer, cumulative time spent traveling and at activities taking into account changes in employment (counter value)
atActivity – Boolean, true= stationary, false = moving, default value = true
atRisk – Boolean, true = vulnerable to being victimized, false = safe, default value = false.

timeCounter – integer, keeps track of the cumulative time at an activity but is reset when activity type changes or agent begins to travel. Default value = 0.
wealth - integer, the amount of wealth an agent has
position – the position in the pathNodes array that an agent is occupying
moveStatus – boolean, true = traveling, false = not traveling, default value = false
occupiedNode – ActiveNode, each of these nodes have at least one civilian agent on them.
empHome – integer – the home node for an agent while employed
empMain – integer – the main node for an agent while employed
empRec1 – integer – the first recreation node for an agent while employed
empRec2 – integer – the second recreation node for an agent while employed
unempHome – integer – the home node for an agent while unemployed
unempMain – integer – the main node for an agent while unemployed
unempRec1 – integer – the first recreation node for an agent while unemployed
unempRec2 – integer – the second recreation node for an agent while unemployed
unempPathNodes – java.util.ArrayList – the list of nodes that an unemployed agent traverses during the course of a day
empPathNodes– java.util.ArrayList – the list of nodes that an employed agent traverses during the course of a day
changeEmpStatus – Boolean, true = change the employment status, false= do not change employment status, default value = false
numVict – integer, number of times an agent get robbed, default value is 0
numOffen – integer, number of times an agent commits a robbery, default value is 0
totTimeTraveling – integer – cumulative travel time even with employment changes (counter)
totTimeExposed – integer – cumulative 'at risk' for street robbery (i.e. not at home)
timerHome – integer – cumulative time spent at home over the course of the model run
timerMain – integer – cumulative time spent at main over the course of the model run
timerRec1 – integer – cumulative time spent at rec1 over the course of the model
timerRec2 – integer – cumulative time spent at rec2 over the course of the model
timerRepeat – integer – cumulative time spent unable to offend until the REPEAT value is reach
location – Place, is the street node object as a place
strnode – integer, holds the strnode id number
employmentStatus – boolean, employed = true and unemployed = false. Default value =true.

Actions that are in Schedule

step – at every 1 tick
payCitizens – at interval of 20,160 ticks

### ***Name: ActiveNode***
### ***Group Name: activeNodes***

The ActiveNode class exists as a computational device to avoid having to check all the places at each step. At each step, the nodes where citizens and cops are located are associated with the ActiveNode class. This limits the maximum number of nodes that would have to be checked to 1,000 (i.e. the maximum number of nodes if each agent was at a unique node) instead of 16,035.

Actions:

None

Fields:

strnode – integer, has the node number of the agent
agentList – java.util.ArrayList, has the list of agents at the node

Actions in the Schedule

None

### ***Name: Cop***
### ***Group Name: cops***

The Cop agent class is used to increase the risk of committing a crime (i.e. increase the guardianship at a place). Cops cannot be victimized nor can they commit a crime. Their movement patterns are random. They move from their current place to a randomly chosen adjacent place at every model tick by consulting the myNeighbors field in the Place class which lists all the neighbors for their current node.

Actions:

step() –

First the action gets the list of all the places in the Place class. For each node that has a cop, the list of neighbor nodes is shuffled and then the cop is assigned to the first position in the node list. The strnode and the location fields are changed to reflect the cop's new position. The location field is a Place object which allows the decideRob() action to see if there are cops at the active node. Cops move last in each tick.

Fields:

strnode – integer, has the node number
location –Place, is the street node object as a place

Actions in the Schedule

step() – runs at every 1 tick.

## 3.4 Other Model Components

A sequence graph is included that tracks the number of times a street node is visited by any agent, the number of times a robbery occurs on the node and the number of times a robbery is deterred by the presence of a cop.

***Name:*** ***_Tracking***
***Title:*** ***Activity Graph***

Series:

totalRobberies – cumulative number of robberies at a node
totalDeterred – cumulative number of robberies prevented by a cop at a node
totalIntersect - cumulative number of times more than one agent is at a node

Schedule:

Update of graph is run on every 100 ticks.

## 3.5 Random Number Distributions

The use of robust random number generators (RNGs) is essential to producing high quality, scientifically defensible results. This simulation uses the Mersenne Twister RNG for all the random numbers in the model. Each of the random number distributions used relies on the same seed. The following section describes all the random number distributions used in the model, how they are created and which action calls them.

**Actions That Use RNG'S**

**initCitizens()**

Uniform random number distribution that is used to choose subsets of agents from the totals set for criminal propensity, employment status, number from between 0 and the number of agents in the model. Random.nextIntFromTo(0,self.NUM_Places - 1)

Normal random number distribution that is used to assign the amount of time to spend at home with a mean of the experimental value and a standard deviation of 10 percent of the mean (very peaked distribution) as is shown below.

```
societyPercHome = 1 - self.SOCIETAL_TIMEAWAY
  standardDeviation = (self.MODEL_DAY * societyPercHome)*.10
  meanTimeHome = self.MODEL_DAY * societyPercHome
  Random.createNormal(meanTimeHome,standardDeviation)
```

Normal random number distribution that is used to assign wealth to the agents:
Random.createNormal(self.WEALTH_MEAN,self.WEALTH_SD)

---

**decideRob()**

Uniform RNG to represent differences in perceived guardianship:
Random.createUniform(-2,2). This adds or subtracts up to 2 agents to the perceived guardianship value (e.g. a place manager might represent 2 agents, while someone in car might not count for as much):
Random.uniform.nextIntFromTo(self.MIN_GUARDIANSHIP, self.MAX_GUARDIANSHIP)

Uniform RNG to represent differences in perceived target suitability:
Random.createUniform(-1,1) – this adds or subtracts one unit of wealth from the highest wealth agent:
Random.uniform.nextIntFromTo(self.MIN_SUITABLITY, self.MAX_ SUITABLITY)

Uniform RNG (uses target suitability RNG above) used to represent the influence of other unknown factors when the decision to offend could go either way based on guardianship and suitability (randDecision):
Random.uniform.nextIntFromTo(self.MIN_SUITABLITY, self.MAX_ SUITABLITY)

---

**initCops()**

Uniform RNG to choose place index numbers to which to assign cops. Randomly chooses from the entire set of street nodes.
Random.uniform.nextIntFromTo(0, self.NUM_PLACES – 1)

---

## 3.6 Statistics Files

1) C:/model_output<condition>/modelRunDatav1.csv – in the initCitizens() write out agent values for criminal propensity, employment and wealth; in the initModel() write out parameter values for model
2) C:/model_output<condition>/citizen<agentName>.csv – a series of files (one for each agent) that are written from the writeCitizenTravelInfoFiles(). Each file contains the: tick, the timers for home, main, rec1, rec2, time exposed and time traveling, the position of the agent in their path, number of offenses, number of victimizations and the amount of time originally assigned to spend at each place.
3) C:/model_output<condition>/statistics.csv – writes out the times actually spent at activity nodes. All agents are in one file. Easy to get percentages of time spent at home, main, rec1, rec2 from this file.

4) C:/model_output<condition>/occupiedSnapshot<Tick>.csv – a series of files are created (one at each tick). Documents the distribution of agents across the streetnodes and keeps track of how many agents were at any one street node and which agents were there.
5) C:/model_output<condition>/timeAtActivityNodes.csv – file that has the amounts of time assigned for activity nodes. All agents are in one file.
6) C:/model_output<condition>/citizenChar.csv – prints every 60 ticks. Has information to monitor the model level variables. Files is created by createCitizenTravelOutputFiles() and written to in dataRecorder()
7) C:/model_output<condition>/path<agentName>.csv – creates and writes to the files to hold the list of nodes that each citizen visits during random movement. Action must be scheduled to run.

*Appendix 2: Street Robbery Model Documentation: Temporal and Activity Space Versions*

# Street Robbery Model Documentation: Temporal and Activity Space Versions

This documentation explains the versions of the Full Street Robbery Model from a programming perspective. The documentation follows guidelines for sharing simulation research in Axelrod (Forthcoming). A general description of the model is provided in the main text. The criminological basis for the model is provided in Groff (Forthcoming-a). The theoretical basis for the representation of activity spaces is discussed in Groff (Manuscript available from author). The implementation of agent movement on a vector network is detailed in Groff (Forthcoming-b). The model is created in Agent Analyst.

## 1. REPAST PY AND AGENT ANALYST

RePastPy was developed by Argonne National Laboratories to provide a Python-based syntax for rapid model development. The software uses a hybrid language that has been dubbed Not Quite Python (NQPY). NQPY uses Python-syntax to access Java classes. Agent Analyst is an extension to RePastPy that can be used as a toolbox in ESRI's ArcGIS software. Agent Analyst provides the ability to use data from a geographic information system (GIS) in agent-based models. Public releases of final versions of software are available at http://repast.sourceforge.net/download.html. For general information on RePast please see http://repast.sourceforge.net/index.html. The Street Robbery Model presented here was developed with a beta version of Agent Analyst which utilized Java 1.2.4_06, Python 2.3 and ArcGIS 9.1. The documentation assumes some familiarity with these languages and software products.

## 2. OVERVIEW OF THE MODEL

The versions of the Street Robbery model discussed here are based on routine activity theory (RAT) (Cohen & Felson, 1979) and create a simulation of how an individual agent's decisions on whether or not to commit a street robbery translate into macro-level crime patterns. RAT identifies four elements necessary for a crime to occur. Specifically, the *routine activities* of individuals determine which individuals are at the same place, at the same time. For a crime to occur there has to be a *motivated offender*, a *suitable target* and the lack of *capable guardians*. Each of these elements is represented in the model. A series of experiments are conducted to test whether the outcomes from the model match what the theory predicts (i.e. whether crime will increase as people spend more time away from home). This prediction is tested by systematically changing the amount of time that the society of agents spends away from home. For example, in the first of five conditions agents spend 30% of their time away from home. The results from this model are compared to societies in which agents spend 40%, 50%, 60%, and 70% of their time away from home.

There are two types of people in the model, civilians/citizens and police/cops.[50] Civilians have characteristics describing their employment status, activity status, wealth, and criminal propensity. Two-hundred of the one thousand civilians are assigned criminal propensity; they evaluate each situation for the potential to commit a street robbery considering the level of guardianship (formal and informal) and the suitability of the target. Civilians with criminal propensity can take on the role of offender, target, or informal guardian in any situation; while those without can only play the roles of target or guardian. Police have only one role, formal guardians.

The two versions of the model described here are identical except for the spatio-temporal constraints placed on the civilian's schedules. Police agents patrol randomly without stopping in both versions. Both versions rely on empirical data to inform the movement of agents (i.e. the street network of Seattle, WA). However, civilians in the Temporal version move randomly but follow a time schedule. At the start of each day, citizen agents remain at home for their assigned time and then begin traveling randomly for the rest of each day. They begin the next day at the ending point of the previous day.

In the Activity Space version, each agent has both spatial and temporal constraints on their activities. Civilians follow the same temporal schedule as they did in the Temporal version but also have a set of locations they must visit each day. In this way, an agent's activity space consists of a set of places and the time to stay at each one. The civilians begin each day at their home and travel in a ring pattern among their assigned activity nodes. The Activity Space version uses empirical data to inform both the locations of the places visited and the route taken among those places (i.e. the activity spaces). Section 3 offers a complete description of the technical aspects of implementing activity spaces.

## 3. ACTIVITY SPACES

### 3.1 Implementing Movement Along a Street Network

Agent Analyst does not support the connections to a geodatabase or a network dataset which enable routing in ArcGIS. Consequently, there can be no dynamic routing of agent travel in the model. The alternative strategy to enable directed agent movement among a set of locations was to use GIS functionality to create predefined activity spaces outside of Agent Analyst. Street intersections (represented as points) rather than the street segments are the Places in the model. Thus, movement takes place from street intersection to a connected street intersection rather than from one street

[50] The terms civilian and citizen describe the population of the city who are not police/cops. The general term describing the population was changed to civilian after the program was written and the code has not been changed to reflect that evolution. The same situation is true for the use of the terms police and cop both of which refer to law enforcement officers.

to another. This allows the agent paths to be represented by the series of street intersections that are traversed to visit all four nodes. The above solution also facilitates the dynamic random movement that is required according to the model specification. Random movement is used by the cop agents in both the Temporal and Activity Space versions and by the civilians in the Temporal version of the model.

The extensive data manipulation that was necessary to prepare data for use in the model is diagrammatically represented in Figures 1 and 2. Figure 1 provides an overview of the entire data flow and Figure 2 offers the details of the process of creating activity spaces. Figure 2 should be read left to right and top to bottom. The symbols appear in a legend in the lower right hand corner. They represent different types of files and programs. The rounded boxes represent GIS layers (dark green are lines and light green are points and yellow are polygons). The orange rectangles are files (usually .csv). The bright blue pages are programs.

Figure 1 provides a graphical view of the general data flow. For agent movement, both versions rely on the streets of Seattle represented as intersection nodes. So the movement box on the left describes the data for random movement. The Activity Space version of the model employs GIS data describing streets, blockgroups and zip codes in Seattle to provide the geographic structure within which the citizen and cop agents go about their daily lives. Data describing the geographic distribution of residents, jobs and potential activities for these areas was collected to aid in the development of the agent's activity spaces. To enable dynamic random movement in both versions of the model, a Python script was developed to identify the neighbor nodes for each node in the network. Output data are written to both the street node shapefile and a series of text log files as the model runs.

Figure 1: Data Flow

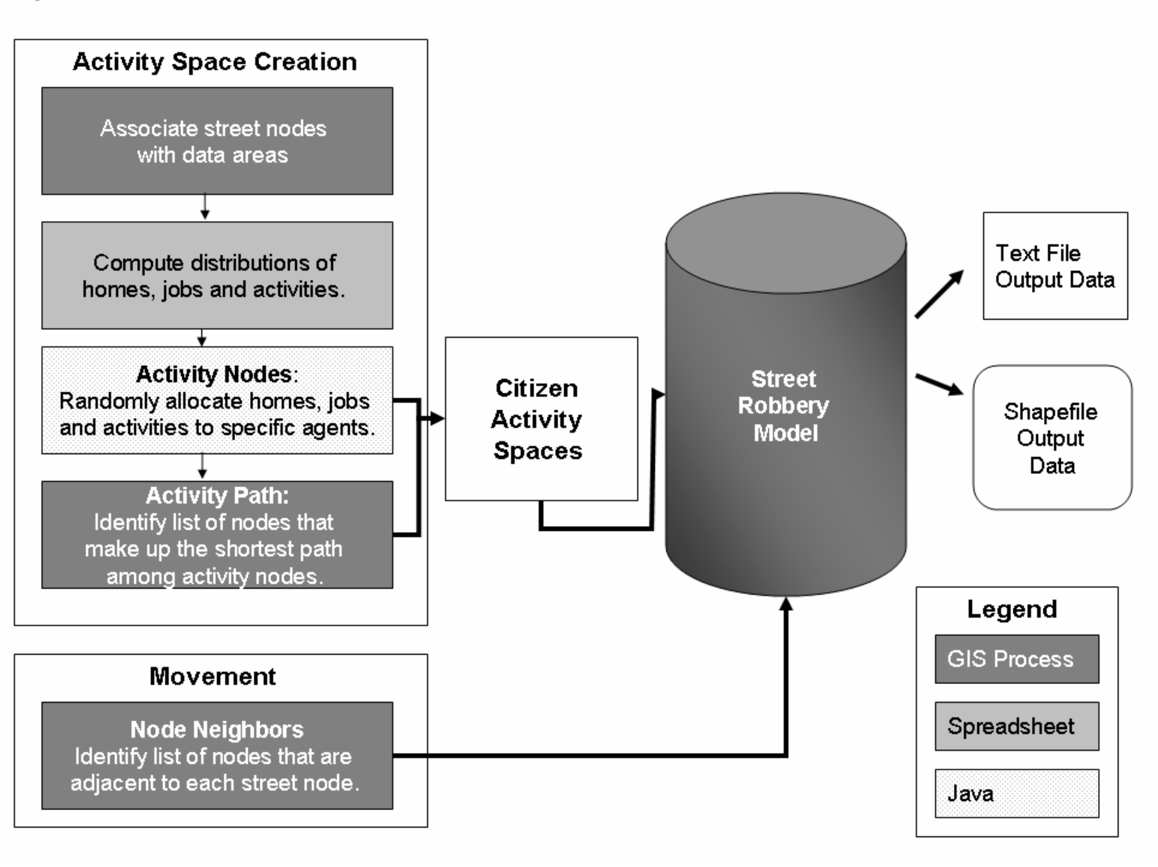
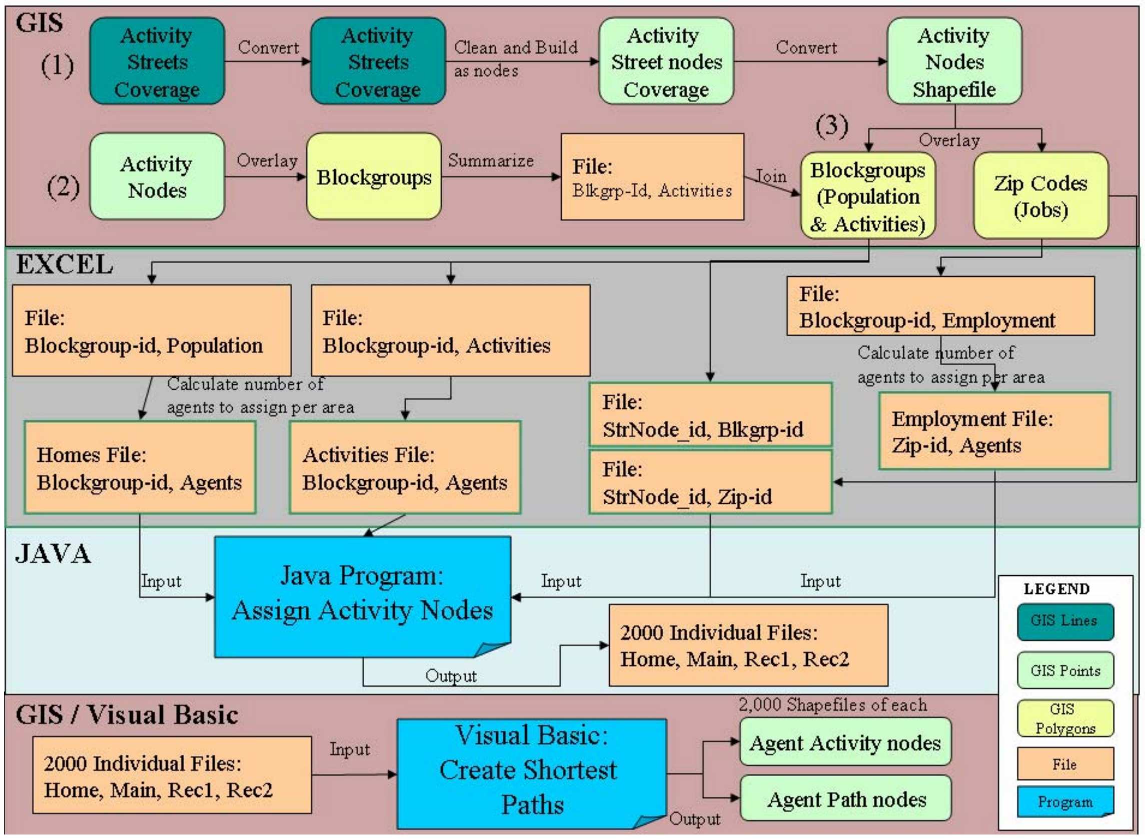

Activity Space Creation
Associate street nodes with data areas
Compute distributions of homes, jobs and activities.
Activity Nodes: Randomly allocate homes, jobs and activities to specific agents.
Activity Path: Identify list of nodes that make up the shortest path among activity nodes.
Citizen Activity Spaces
Street Robbery Model
Text File Output Data
Shapefile Output Data
Movement
Node Neighbors Identify list of nodes that are adjacent to each street node.
Legend
GIS Process
Spreadsheet
Java

Figure 2 provides more detail regarding the type of data that were used to inform the model, how those data were manipulated and by what software. Two types of street layers were used in the study, travel streets and activity streets. First, all linear features in the King County Street Network Database (SND) that were not transportation-related were removed to create the travel streets.[51] Travel streets consist of those streets that can be used for transportation and serve as the basis for cop and citizen movement. Activity streets contain only those streets along which an agent could live, work or undertake an activity (i.e., freeways are excluded). The activity streets were created by removing the freeways from the query used to produce the travel streets.[52] The node layer (activity nodes) was used to allocate the agent home, employment and activity places.

[51] Travel streets consist of the following types of linear features: Streets (code=0), Divided Street (code=1), Parks (Unlimited access) (code=2), Freeway (code=4), Alley (code=6), Parks (Limited access) (code=7), Other agency (code=8), Stairs (code=20), Walkway (code=21), Multipurpose Trail (code=22), Private Street (code=40), Dock (addressable slips) (code=51).
[52] Features along which there could be no employment or housing or activities (i.e., Freeways) are removed leaving the following types of features: Streets (code=0), Divided Street (code=1), Parks (Unlimited access) (code=2), Alley (code=6), Parks (Limited access) (code=7), Other agency (code=8), Stairs (code=20), Walkway (code=21), Multipurpose Trail (code=22), Private Street (code=40), Dock (addressable slips) (code=51).

Additional processing was necessary to create a layer that accurately represents street intersections/nodes. First, both street layers were converted to ArcInfo coverages and the pseudo nodes removed.[53] The street layers were then converted to point shapefiles where each point represents the intersection of two streets. At the end of the first GIS stage, a layer of travel nodes and activity nodes were intersected with the blockgroup polygons and the zip code polygons to provide a list of the street nodes in each blockgroup and zip code. Before the activity spaces for the citizen agents could be created, the locations of the street nodes had to be linked to the polygon layers because those layers contained the data about the potential activity nodes that exist in each area (i.e., blockgroup or zip code).

Figure 2: Creating Activity Spaces for Citizen Agents



GIS
(1)
Activity Streets Coverage
Convert
Activity Streets Coverage
Clean and Build as nodes
Activity Street nodes Coverage
Convert
Activity Nodes Shapefile
(2)
Activity Nodes
Overlay
Blockgroups
Summarize
File: Blkgrp-Id, Activities
Join
(3)
Overlay
Blockgroups (Population & Activities)
Zip Codes (Jobs)
EXCEL
File: Blockgroup-id, Population
File: Blockgroup-id, Activities
File: Blockgroup-id, Employment
Calculate number of agents to assign per area
Calculate number of agents to assign per area
Homes File: Blockgroup-id, Agents
Activities File: Blockgroup-id, Agents
File: StrNode_id, Blkgrp-id
File: StrNode_id, Zip-id
Employment File: Zip-id, Agents
JAVA
Input
Java Program: Assign Activity Nodes
Input
Input
Output
2000 Individual Files: Home, Main, Rec1, Rec2
GIS / Visual Basic
2000 Individual Files: Home, Main, Rec1, Rec2
Input
Visual Basic: Create Shortest Paths
2,000 Shapefiles of each
Agent Activity nodes
Agent Path nodes
Output
LEGEND
GIS Lines
GIS Points
GIS Polygons
File
Program

## 3.2 Creating Activity Spaces

Figure 2 also offers a detailed view of all the stages involved in assembling the citizen agent activity spaces. The first stage, just described, used GIS to: 1) create a layer of street nodes (intersections); 2) assign area identifiers to each street node and 3) assemble data describing population, jobs and activities. The next stage employed

[53] Pseudo-nodes are nodes remaining in the layer from when the SND contained more lines but that no longer represent the intersection of two lines. Failing to delete the pseudo nodes would artificially inflate the number of street intersections in Seattle. Nodes that represent false intersections (e.g. overpasses) and nodes that were not connected to the rest of the network are also deleted to make the node network better represent the actual number of intersections in Seattle.

data from the first stage to calculate the number of citizen agent homes, jobs and activities that should be allocated to each area in the same proportion as they are found in Seattle (e.g. if 10% of the population lives in a particular blockgroup then 10% of the agents are assigned to that blockgroup). Stage three made use of a java program to randomly select a set of activity nodes for each agent. Four activity nodes were selected; those nodes represent a home node, main node (could be work, school etc.) and two recreational nodes (i.e., retail store, gym, coffee shop). Together the four different activity nodes constitute each agent's activity space. The following paragraphs provide a more detailed description of the processes involved in creating activity spaces, regardless of stage.

*Home Node Assignment*
Agent homes are allocated by a multi-step process. First, the total population of each census blockgroup is collected from the 2000 Census of Population and Housing. Seattle had a total population of 564,945 in 2000. That means that each agent in the model represents 565 people who live in Seattle. Next, the percentage of Seattle's total population that lives in each blockgroup is calculated and multiplied by the number of agents (1000) that are in the model to get the number of homes that should be assigned from that blockgroup. At this point two files exist; one file contains the activity node number and the blockgroup in which it is located. The other file contains the blockgroup and number of agents to be assigned a home node from that blockgroup. The same basic methodology is then used to assign work places and activities.

*Employment Node Assignment*
A data source for the number of employees per blockgroup is needed in order to assign the work places of the agents when they are employed. Unfortunately, the number of employees is not available by blockgroup, only by zip code. There far fewer zip codes (n=56) than blockgroups (n=570) in Seattle. Consequently, the employment data is less precise than the blockgroup data (i.e., the units to which employees are assigned are much larger and thus the potential for allocating agents in a way that is not reflected by the actual distribution of employment is higher). The rest of the process is the same as for assigning homes.

*Activity Node Assignment*
The same strategy used for homes and workplaces is also used for activity nodes. Data regarding retail establishments and service establishments in Seattle is used. Retail, entertainment and service businesses classified as the following SIC codes are included in the analysis: 52; 53; 54; 55; 56; 57; 58; 59; 72; 7991; 7992; 7993; 7997; 7999; 82; 83; 84; and 8661. Using a spatial join, each activity is assigned to a blockgroup and then summarized to obtain the total number of activities per blockgroup. These data indicate a total of 18,024 qualifying establishments in the city. Once again, the percentage of total activities in Seattle is calculated for each blockgroup and written out to a file. The file contains two fields; blockgroup-id and the number of activity nodes to be allocated to this blockgroup.

A java program randomly assigns agent homes, work places and activities in the same proportion as they are found in Seattle.[54] In general, the program reads in the three files that describe how many agents should live, work and recreate in each area (i.e. blockgroup or zip code) and the two files that match each street node with the blockgroup or zip code in which it is located. An array of 1,000 numbers is created to assign work places and homes. A separate array of 3,000 numbers is created to use with activity nodes.[55] Those lists are shuffled and then used to select the nodes in random order for home, work and activities. Two thousand files are written out; one for each agent when employed and another for each agent when unemployed. The home, work, rec1 and rec2 are written to the file for a particular agent when employed. The same home, rec1, and rec2 are written to agent's unemployed node set with the addition a new node to replace work.

*Establishing the Path Among Activity Nodes*
The final stage in the process involved finding the shortest path among the activity nodes. However, the standard routing algorithm identifies the streets that are traversed, not the street nodes. This was accomplished via ArcGIS Network Analyst and the process automated via Visual Basic.[56] This is done for both the activity space when employed and when unemployed. The program reads each file of agent activity nodes and uses the X, Y coordinates to convert the activity nodes to a shapefile. The program then calculates the shortest path using a network dataset.[57] The travel nodes that are traversed while traveling the shortest path are written out to agent path files (two for each agent; employed and unemployed). The 4,000 output files describing the activity nodes and activity path nodes for each agent are read into the model and used to define citizen agent movement.

## 3.3 Java Program Documentation
All the java programs are part of a package called AssignNodestoAgents. The two main programs are NodeAssignment and AssignNodeSets. Four additional methods are called upon by the two main programs. The purpose and inner workings of each are described below.

*NodeAssignment()*
NodeAssignment is the main program to randomly select the streetnodes to be part of agent activity spaces. It reads two types of files – 1) nodes per area (blockgrpid, streetnode, x, y) or (zip, streetnode, x, y) that list all the streetnodes in each area and 2) a file that lists each area and the number of agents who live, work or recreate there (area, number of agents). The program writes out three files allHomes, allJobs and allAct.csv that contain a list of 1,000 randomly selected home and work nodes, and

[54] Detailed documentation on the java program is available in section 3.3.
[55] Three thousand nodes are needed because an activity space requires a) 1,000 nodes for recreation place 1 (rec1), b) 1,000 nodes for recreation place 2 (rec2) and 3) 1,000 nodes to replace the Main node when the agent is unemployed.
[56] The custom program was created in Visual Basic and run in ArcGIS 9.1 to identify the street nodes traversed by each agent. Section 3.4 has the documentation for the program.
[57] The network dataset is created from the travel street network and is required to be able to use Network Analyst.

3,000 activity nodes. The program has two methods, collectNodes and selectRandomNodes. The collectNodes reads a csv file of records containing the blockgroup/zip code, streetnode, x and y for all streetnodes in Seattle. This method is essential because it enables the identification of a set of streetnodes for each blockgroup from which the specified number of homes or jobs can be selected. The selectRandomNodes method actually creates the set of randomly selected streetnodes. It begins by reading the file with the area-id and number of streetnodes to choose (homes, jobs, and activities). It then generates a set of random numbers. Then a number is chosen from the set of random numbers that is between zero and the size of the BlkgrpSet set for that blockgroup. The program then checks that number against the index numbers of the streetnodes and selects the streetnode in an area that has the corresponding index number. These actions are repeated until the correct number of nodes to be assigned to that area is reached.

*AssignNodeSets()*
AssignNodeSets reads the three files generated by NodeAssignment and creates two sets nodes of each of the 1,000 agents (employed and unemployed). Since the nodes in the files were randomly chosen but are still grouped by area-id, a method is needed to "shuffle" the locations before assigning them to the individual agent's activity spaces. Since I could not find a way to shuffle the files, instead two arrays are created (one with 1000 numbers and the other with 3000 numbers). These lists are shuffled and the numbers used to choose the indexes of the streetnodes in homes, jobs, and activities. The streetnodes are written out so that each agent has the same home, act1 and act2 nodes between their two paths. For employed they have home, work, act1 and act2. While unemployed, agents have home, act1, act2 and act3. The program writes out 2000 uniquely named files each with a streetnode, x-coord and y-coord. These files are then used in ModelBuilder to generate the pathnodes (each intersection along the path among the nodes in the activity space).

*StrNodeLoc ()*
StrNodeLocs are objects that represent the streetnode number and x,y of a particular streetnode. Creating an object that represents these associated pieces of information, I could keep the three variables (streetnode number, x-coord and y-coord) together and easily write them out to files.

*BlkGrpNodes()*
BlkgrpNodes are objects that represent the streetnode number, x-coord and y-coord of all the streetnodes in a particular blockgroup (actually it is a set of StrNodeLoc objects). Nodes associated with a blockgroup are selected from the overall list and put into a collection (in this case a vector). There is a method called collectInfo in the NodeAssignment that reads a csv file of records that give the blockgroupid, streetnode, x and y for each record. This method is essential to creating a set of streetnodes for each blockgroup from which the specified number of homes or jobs can be selected.

*RandomNodeSet()*
RandomNodeSet objects are structured the same as BlkGrpNode objects but they represent a set of randomly selected streetnodes (streetnode number, x,y). Once again it is a set of StrNodeLoc objects. There are three types of RandomNodeSets, home, job, and activity.

*Mersenne()*
This implements the class that uses the Mersenne Twister algorithm for random number generation so that the Mersenne Twister random number generator (RNG) can be used directly. However, the tutorial for Random Numbers on the Repast website states the Random class in Java also uses Mersenne Twister as the RNG. Consequently, the direct call implemented here is not necessary.

<u>3.4 Visual Basic Program Documentation</u>
This stand-alone application allows you to select a folder containing comma-delimited ASCII files.[58] In this case, the files contain the activity nodes for the civilian agents. It reads the activity nodes (4 nodes), puts the nodes on a street network, and calculates the shortest path route between each set of nodes in a ring pattern. Next, the resulting route is joined back to the junctions in a network dataset based on the matched records so the only records selected are the ones that were traversed. Then the nodes are written out to a shapefile, two routes for each agent (employed and unemployed)

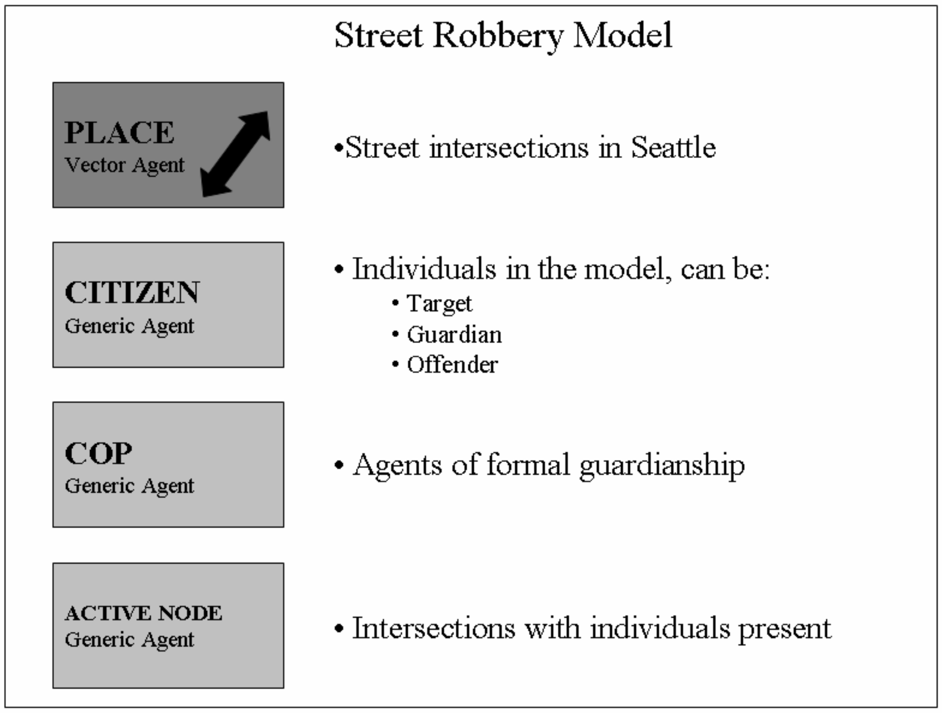
The visual basic project for the application consists of a basutil.bas which contains several subroutines that enable the above. The form is called frmAgentPaths.frm and provides the graphical user interface. The prjAgentPaths.vbp is the visual basic project file that is needed to open the project. The form uses the ArcObjects MapControl, one property is the ability to open a map document that references the network dataset and junctions that are to be used.

## 4. MAIN MODEL AND AGENT CLASSES

There is one main model and four classes of agents in each of the versions (Figure 3). Three of the classes consist of generic agents – cops, civilians and active nodes. One class is made up of vector agents, places. Vector agents have an inherent spatial property that is needed in the simulation, generic agents do not. Each of the classes has a set of actions, a schedule that controls when the actions run and a set of fields. The actions control a variety of functions necessary to the running of the model including agent initialization, agent movement and agent decision-making. The actions can be scheduled to run at each model tick, at a specified interval or just once during the course of a model run. The fields contain the data that are available to describe the members of each class. There is also a Sequence Graph that is used to graphically display the number of robberies as they occur during the model run.

[58] The author gratefully acknowledges Jo Fraley for writing and making this program available to subsequent researchers.

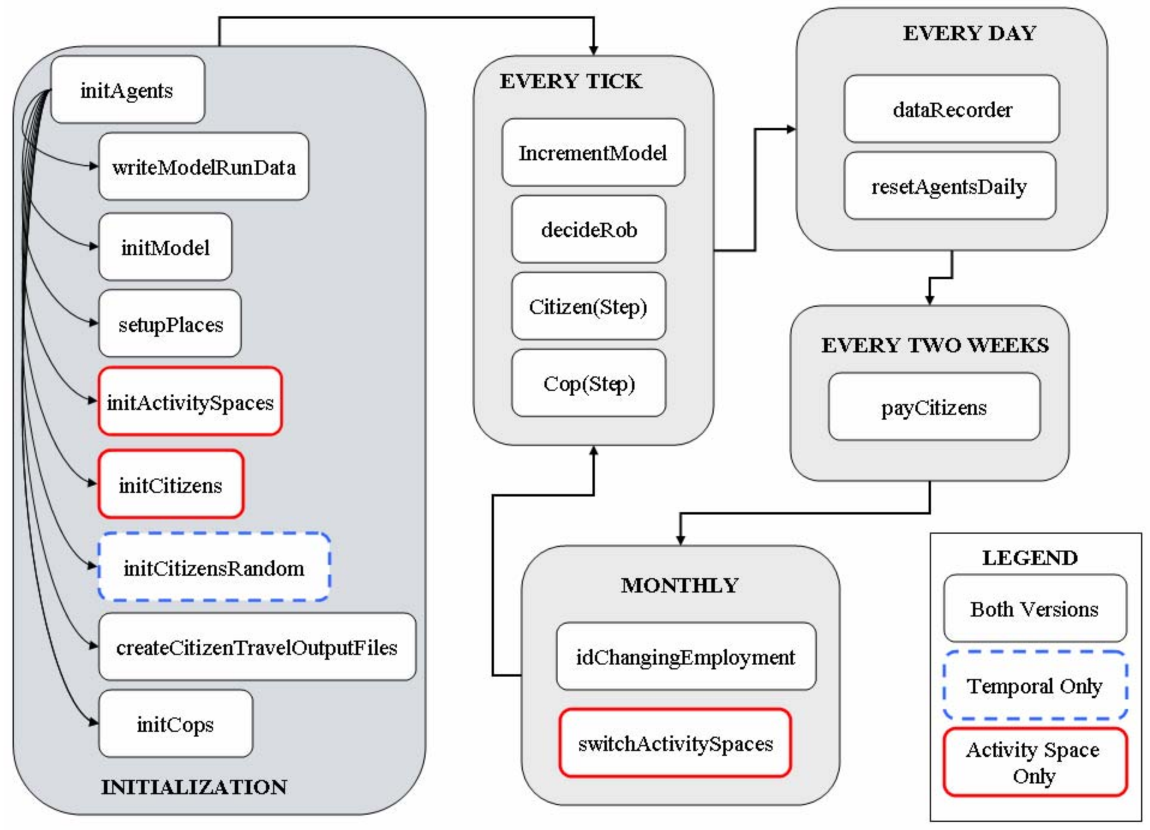
Figure 3: Street Robbery Model Classes


Street Robbery Model
PLACE
Vector Agent
•Street intersections in Seattle
CITIZEN
Generic Agent
• Individuals in the model, can be:
• Target
• Guardian
• Offender
COP
Generic Agent
• Agents of formal guardianship
ACTIVE NODE
Generic Agent
• Intersections with individuals present

## 4.1 The Main Model

The main models for each version are called StreetRobTemporal and StreetRobActivitySpace. They have display names of Street Robbery Temporal and Street Robbery Activity Space. The main model contains all the actions for the initialization of the model. Each of the actions is listed and their function is explained below. The text describes the Activity Space version and differences for the Temporal version are noted in parentheses where they occur. Fields that are in all capital letters are static/global variables that can be called from anywhere in the models. The main models are also where all the random number distributions are created so they can be called during the appropriate parts of the model run. Please see Figure 4 for a graphical representation of the order of execution for the actions in the models.

Figure 4: Order of Action Execution in the Model



<u>Actions:</u>

initAgents()

This action is the first action to run in the model. As such, it is also the main action that calls other actions to initialize the agents. It calls the following actions for both versions: writeModelRunData(); initModel(); createCitizenTravelOutputFiles(); setupPlaces(); and initCops(). For the Temporal version it calls initCitizensRandom() and for the Activity Spaces version it calls initCitizens() and initActivitySpaces(). Specifies a random number seed and creates the uniform random number distribution and a normal distribution with the specified seed. The action also creates the street message display function (that is currently not used).

updateDisplay()

The updateDisplay action changes the display in ArcMap. Execution of this action is done through the schedule.

writeAgents()

This action writes the agent values from the place class to the shapefile. The action runs before the updateDisplay() action. The symbolization settings are created in the properties of the strnodes2 layer in ArcGIS. The same shapefile is added into the .mxd file twice; once to symbolize the total robberies at a place, another time to symbolize the total number of visits to a place.

setupPlaces()

The primary function of setupPlaces is to initialize a linked hashmap and identify the neighbors of each place for random movement. First, a linked hashmap is created to store a list of all the places with strnode-id as the key. Next, the action reads the nodeNeighbors.csv file and associates the set of neighbors with the correct Place (i.e., it populates the field myNeighbors in the Place class).

showMessage()

The showMessage() action enables the display of custom messages to an output window.

incrementModel()

This action is scheduled to execute at every tick of the model. The action does the following: 1) increments model counter, 2) calls the writeOccupiedNodes() action, 3) calls decideRob(); 4)clears the agent list associated with the ActiveNode class and increments the criminal agent's time to reoffend counter. It also has code to call the writeOccupiedNodes() and writeCitizenInfoPaths() methods to track agent movement.

initModel()

The initModel() is called by initAgents() and sets values for constants and static variables in the model some of which are parameters and can be changed through the RePast GUI at model run time.

initActivitySpaces() (not used in StreetRobberyTemporal)

Reads the files of activity nodes and paths and creates a new Civilian agent with an activity space as specified by the appropriate activity nodes and pathnodes. Uses the assignNodeInfo() action in Citizens to set the field values in Citizen agents.

initCitizens()(not used in StreetRobberyTemporal which uses initCitizensRandom instead)

Sets criminal propensity, wealth, employment, and assigns values to the time to stay away from home fields using the random number distributions created in initAgents(). Action also writes out timeHome, timeMain etc. fields to a file for validation. The times are calculated by starting with the time to spend at home (because this has to be allocated so that the average is a certain number). Next, the number of pathnodes the agent has to traverse in a day is accounted for. Then a check is done to make sure an agent does not have less time left then they are required to stay at home. If that statement evaluates to true, then another random time is assigned to that agent and the statement is reevaluated until there is time left to do other activities. Next, the remaining time is divided in half and half is assigned to the Main activity node. The remaining time is multiplied by a randomly chosen fraction between .10 and .90 and the resulting value is assigned to Rec1. The Rec2 activity node is assigned the remainder of the time.

initCitizensRandom() (not used in StreetRobberyActivitySpace)
Sets criminal propensity, wealth, employment, and reads the activity schedule used in the StreetRobberyActivitySpace version from files. Action also writes out timeHome, timeMain etc. fields to a file for validation. The times were calculated during the run of the StreetRobberyActivitySpace model so they are identical between the two models. A new field is created to divide the time traveling into three equal parts. The agent uses these times to randomly travel before stopping at its next activity. In this way, time spent at activities is separated by time travel.

decideRob()
Contains the code to evaluate who is at the occupied nodes and then decide whether a crime should occur. First, the action checks to find out which agents are at a streetnode by using the ActiveNodes class. Next, the presence or absence of official guardianship (in the form of a cop) is evaluated. If a cop is not present, then the program continues to evaluate the civilian agents at a node. The list of agents at the node is randomly reordered so the same agent is not evaluated first each time and if there are two offenders at a node, each has a random chance of getting to decide to commit a crime.[59] Then each civilian agent is checked for criminal propensity. The first agent found becomes the criminal decision-maker for this situation. Program also takes into account whether the agents at the node are 'at risk' (which is set in Citizens.step()). Next a target agent is identified by checking the wealth of each agent. The agent with the most wealth is chosen to be the target but only if their wealth is greater than or equal to the criminal agent's wealth. Only nodes with two or more agents and an offender are evaluated as far as the actual decision to commit a crime. The suitability and guardianship terms are calculated and used in the decision to commit a crime. If a crime occurs, the action changes the following field values: victim and offender wealth, number of victimizations and total robberies at the place. The decideRob() action is called by incrementModel().

[59] Although this method required additional coding, the use of randomly generated numbers ensures that the same result will be achieved each time a run is conducted with the same random number seed. The shuffle method cannot be used because the order of the agents changes each time the action is given regardless of the random number seed.

writeOccupiedNodes()

This action writes out the distribution of agents across street nodes for diagnostics and data analysis. The action is called by the incrementModel() action so it occurs at every tick (not currently used). However, it could be written out less frequently by scheduling the action instead.

idChangingEmploymentStatus()

This action is scheduled to run every two weeks. The action identifies the 3% of agents whose employment status will change by setting the field changeEmpStatus to true.

switchActivitySpace() (not used in StreetRobberyTemporal)

Checks to see if agent has been identified to switch employment status (by looking at the field – changeEmpStatus that is calculated in idChangingEmploymentStatus()) and then calculates new times to spend at activity nodes just for those agents who change. Uses the same methodology as in the initCitizens() action to allocate time spent at activity nodes. Last, the action updates the field values of main, rec1, rec2 and pathNodes to reflect the new path (employed or unemployed).

initCops()

Creates cop agents and assigns them to a strnode (number) and a location (Place). Action uses a uniform distribution to select the nodes on which to place the cops at the start of the model.

resetAgentsDaily()

At end of the model day, all agents are reset to be back at home so they are at an activity node (home), not at risk and not moving. In the Activity Space version this is necessary because the agents travel a random number of street nodes during each turn.

createCitizenTravelOutputFiles()

Creates two different types of output files to which citizen data can be written. One creates a unique file for each citizen agent to which output can be written by the writeAgentInfoFiles() action (output/citizenX.csv). The other type of file is a single file to which data can be written at specified intervals to monitor societal-level citizen characteristics (output/citizenChar.csv). This file is written to by dataRecorder().

writeCitizenTravInfotoFiles()

Uses the files created by createAgentOutputFiles() and writes out information about the individual agents at different points in the model (not currently called).

writeModelRunData()
This action creates a log file to capture model parameter values, error messages and critical statistics during each model run. Many actions write to the file as necessary.

writeStatistics()
This action captures the final field values for citizen agents pertaining to activity spaces and crime in a single file. Writes out the aggregate time spent at home, main, rec1, rec2, travel, and exposed; the assigned time to spend at home, main, rec1, rec2, travel; and the Total number of offenses and victimizations. These same statistics are written to individual agent files by the createCitizenTravelOutputFiles() and writeCitizenTravInfoFiles() actions. The action writes to output/statistics.csv which is written one time at end of simulation and is very helpful for understanding victimization, offending and wealth for each agent.

dataRecorder()
This action takes the place of the data recorder that I could not get to work in the model. It records variables that change during the model run such as: number of unemployed agents, average wealth, robbery rate, total victims, total repeat victims, total offenders, total repeat offenders, percent exposed, percent traveling, and number of active offenders. Writes to output/citizenChar.csv.

writeFinalAgents()
This action writes out the ending values in the shapefile to a named model output directory for later analysis.

writeCitizenInfoPaths()
For random movement and to verify directed paths, this action provides a convenient way to write out every street node visited by an agent.

Fields:

messageDisplay - uchicago.src.simbuilder.util.MessageDisplay, displays messages while model is running. Parameter
modelStep – integer, counter that keeps track of model steps, 1 minute steps
MODEL_HOUR – integer, number of steps in an hour, 6 x 60 = 360 steps in an hour. Default value = 60.
MODEL_DAY – integer, number of steps/minutes in a day, 24 x 360 = 8,640 steps = 1,440 minutes in a day.
MODEL_WEEK – integer, number of minutes in a week, 7 x 1,440 = 10,080 minutes in a day; 7 x 8640 = steps in a day.
MODEL_YEAR - integer, number of minutes/steps in a year, 365 x 1,440 = 525,600 minutes in a year; 365*8640 = 3,153,600 steps
SOCIETAL_TIMEAWAY – double, Default value = .70,

totRob – integer - Default value = 0, Cumulative number of robberies in the model run. Parameter

totRob – integer, cumulative number of robberies for model.

placeMap – java.util.hashmap – hashmap of strnode-ids

AGENTS – integer – Default value 1000, total number of agents in the model. Parameter

totDeter – integer, Default value = 0, total number of robberies deterred by presence of cop for the entire model run. For this to increment there had to have been more than one agent and a criminal agent at the node.

totIntersect – integer, total times there were more than two agents at a node and a criminal. Represents the number of potential crime situations and is a running total for model run.

NORM_TRAVEL – cern.jet.random.Normal – random number distribution for number of positions to move while traveling

REPEAT – integer, default value is 60-time a criminal has to wait before re-offending. Parameter

COPS – integer, number of cop agents in the model, default value is 200. Parameter.

MIN_GUARDIANSHIP – integer, contains minimum amount of random error in the perception of guardianship by the criminal agent, default value is -2. Parameter.

MAX_ GUARDIANSHIP – integer, contains maximum amount of random error in the perception of guardianship by the criminal agent, default value is 2. Parameter.

MIN_SUITABILITY - integer, contains minimum amount of random error in the perception of target suitability by the criminal agent, default value is -1. Parameter.

MAX_SUITABILITY - – integer, contains maximum amount of random error in the perception of target suitability by the criminal agent, default value is 1. Parameter.

NUM_PLACES – integer, number of street intersections in the model. Parameter.

WEALTH_MEAN – integer, mean of the wealth distribution for all agents in the model. Parameter.

WEALTH_SD – integer, standard deviation of the wealth distribution for all the agents in the model. Parameter.

<u>Actions that are in Schedule</u>

incrementModel – at every 1 tick

resetAgentsDaily - at interval of 1440 ticks (run last)

dataRecorder – at interval of 1,140

idChangingEmploymentStatus – at interval of 40,319 ticks

switchActivityStatus - at interval of 40,320 ticks (run last)

writeStatistics – at end (525,600) ticks

writeStatistics – at interval of 525,599 ticks (Activity Space only)

4.2 Vector Agents
There is only one vector agent class in the simulation and it consists of the set of places (street intersections) in Seattle. To clarify, each street intersection (also called a street node) is a place in the model. There are 16,035 places in the model and they exist in the shapefile called strnodes2.shp that is a point file. This class contains the geographic information about the distribution of agents and robberies (i.e., where the agents and the robberies are located in Seattle).

***Name: Places***
***Group Name: places***

Actions:
None.

Shapefile fields:
ARC_ - integer, internal arc-id used by ArcGIS
STRCL_ - integer, internal node number used by ArcInfo
STRCL_ID – integer, node number used in the model
citiStart – integer, node at which a citizen agent starts the simulation
copStart – integer, node at which a police agent starts the simulation
crimStart – integer, node at which a civilian agent with criminal propensity starts the simulation
the_geom – com.vividsolutions.jts – geometry of point
totPrevent – integer, total number of potential crime situations in which a cop prevented the crime at a node that would have been committed otherwise
totalRob – integer, total number of robberies at a node
totalVisit – integer, total times any agent visited a node

Class Fields:
strcl_ - integer, street node number
myNeighbors – java.util.ArrayList – list of nodes adjacent

Actions that are in Schedule
None

4.3 Generic Agents
Overview of Generic Agents
There are three generic agent classes in the model, citizens, active nodes and cops. These classes are not inherently spatial in nature but through object-oriented programming, the members of the classes can be associated with members of the Place class. Each of the generic classes is described in this section.

***Name:*** ***Citizen***
***Group Name:*** ***citizens***

The citizen class contains all the citizen agents in the model. One of the interesting facets of this implementation is that citizen agents with criminal propensity and those without are modeled exactly the same as far as activity spaces, initial wealth, pay schedule and employment status. The only characteristic that differs is the presence of criminal propensity. Only those citizens who have criminal propensity evaluate criminal opportunities and are able to make the decision to offend. All citizen agents can be victims of street robbery.

<u>Actions:</u>

step() (Activity Space version)

This action controls the movement of the agents along their paths by incrementing their position in their pathNodes field. Keeps track of time (timeCounter) spent at an activity. Changes the atRisk, atActivity and moveStatus variables depending on which activity and whether an agent is traveling between activities. Dynamically creates the ActiveNode class with each step. Only agents at ActiveNodes are evaluated during the decideRob() routine.

step() (Temporal version)

This action controls the random movement of the agents by having them choose a neighbor node to move to. Keeps track of total time spent in their time schedule (timeCounter) throughout the course of a day. Changes the atRisk, atActivity and moveStatus variables depending on the activity of the agent and whether an agent is traveling between activities. Dynamically creates the ActiveNode class with each step. Only agents at ActiveNodes are evaluated during the decideRob() routine.

assignNodeInfo() (not used in the Temporal version)

Assigns the field values from the activity node and path files generated by the Java/GIS programs to each of the Citizen agents (see Figure 2). The following fields are assigned: name, home, empHome, empMain, empRec1, empRec2, empPathNodes, unempHome, unempMain, unempRec1, unempRec2, unempPathnodes and currentNode. Is called by the model action, initActivitySpaces().

payCitizens()

Pay citizens who are employed every two weeks (20,160 ticks).

<u>Fields:</u>

placeNode – Place, associated with vector group of Places
name – string, name of the agent
home – integer, home node of agent

main – integer, work, school or other significant activity node, this value reflects current employment status (emp or unemp)
rec1 – integer, one activity node, this value reflects current employment status (emp or unemp)
rec2 – integer, another activity node, this value reflects current employment status (emp or unemp)
pathNodes – java.util.ArrayList, the nodes traveled to move among activities, reflects current employment status pathnodes
currentNode – integer, holds the strnode_id of node the agent is occupying
criminalPropensity – Boolean, whether or not an agent thinks about committing a crime, default value=false.
timeHome – integer, minutes spent at home
timeMain – integer, minutes spent at work
timeRec1 – integer, minutes spent at recreation one
timeRec2 – integer, minutes spent at recreation two
timeTraveling – integer, minutes spent traveling among activity nodes (assigned as part of activity space), default value=0
atActivity – Boolean, true= stationary, false = moving, default value = true
atRisk – Boolean, true = vulnerable to being victimized, false = safe, default value = false.
timeCounter – integer, keeps track of the cumulative time at an activity but is reset when activity type changes or agent begins to travel. Default value = 0.
employmentStatus – boolean, employed = true and unemployed = false. Default value =true.
wealth - integer, the amount of wealth an agent has
position – integer, the position in the pathNodes array that an agent is occupying
moveStatus – boolean, true = traveling, false = not traveling, default value = false
occupiedNode – ActiveNode, each of these nodes have at least one civilian agent on them.
empHome – integer – the home node for an agent while employed
empMain – integer – the main node for an agent while employed
empRec1 – integer – the first recreation node for an agent while employed
empRec2 – integer – the second recreation node for an agent while employed
unempHome – integer – the home node for an agent while unemployed
unempMain – integer – the main node for an agent while unemployed
unempRec1 – integer – the first recreation node for an agent while unemployed
unempRec2 – integer – the second recreation node for an agent while unemployed
unempPathNodes – java.util.ArrayList – the list of nodes that an unemployed agent traverses during the course of a day
empPathNodes– java.util.ArrayList – the list of nodes that an employed agent traverses during the course of a day

changeEmpStatus – Boolean, true = change the employment status, false= do not change employment status, default value = false
numVict – integer, number of times an agent get robbed, default value is 0
numOffen – integer, number of times an agent commits a robbery, default value is 0
totTimeTraveling – integer – cumulative travel time even with employment changes (counter) , default value is 0
totTimeExposed – integer, cumulative time spent traveling and at activities taking into account changes in employment (counter value) , default value is 0
timerHome – integer – cumulative time spent at home over the course of the model run, default value is 0
timerMain – integer – cumulative time spent at main over the course of the model run, default value is 0
timerRec1 – integer – cumulative time spent at rec1 over the course of the model, default value is 0
timerRec2 – integer – cumulative time spent at rec2 over the course of the model, default value is 0
timerRepeat – integer – cumulative time spent unable to offend until the REPEAT value is reach, default value is 0
location – Place – place object representing physical location of agent
strnode – integer – strnode number of intersection representing physical location of agent
nodeList – java.util.ArrayList – list of nodes agent has visited using random movement
timeEmpHome – integer – time to spend at home while employed (Temporal only)
timeEmpMain – integer – time spent at main while employed (Temporal only)
timeEmpRec1 – integer – time spent at rec1 while employed(Temporal only)
timeEmpRec2 – integer – time spent at rec2 while employed (Temporal only)
timeEmpTraveling - integer, time to spend traveling while employed (Temporal only)
timeUnempHome – integer – time to spend at home while unemployed (Temporal only)
timeUnempMain – integer – time spent at main while unemployed (Temporal only)
timeUnempRec1 – integer – time spent at rec1 while unemployed (Temporal only)
timeUnempRec2 – integer – time spent at rec2 while unemployed (Temporal only)
timeUnempTraveling - integer, time to spend traveling while employed (Temporal only)
travelTimeSplit – integer – travel time between activities (Temporal only)

Actions that are in Schedule
step – at every 1 tick

payCitizens – at interval of 20,160 ticks

## *Name: ActiveNode*
## *Group Name: activeNodes*

The ActiveNode class exists as a computational device to avoid having to check all the places at each step. At each step, the nodes where citizens and cops are located are associated with the ActiveNode class. This limits the maximum number of nodes that would have to be checked to 1,000 (i.e. the maximum number of nodes if each agent was at a unique node) instead of 16,035.

<u>Actions:</u>
None

<u>Fields:</u>
strnode – integer, has the node number of the agent
agentList – java.util.ArrayList, has the list of agents at the node

<u>Actions in the Schedule</u>
None

## *Name: Cop*
## *Group Name: cops*

The Cop agent class is used to represent formal guardianship at a place. The presence of a cop increases the risk of committing a crime (i.e. increase the guardianship at a place). Cops cannot be victimized nor can they commit a crime. Their movement patterns are random. They move from their current place to a randomly chosen adjacent place at every model tick by consulting the myNeighbors field in the Place class which lists all the neighbors for their current node.

<u>Actions:</u>
step() –
First the action gets the list of all the places in the Place class. For each node that has a cop, the list of neighbor nodes is reordered and then the cop is assigned to the first position in the node list. The strnode and the location fields are changed to reflect the cop's new position. The location field is a Place object which allows the decideRob() action to see if there are cops at the active node. Cops move last in each tick.

<u>Fields:</u>
strnode – integer, has the node number
agentList – java.util.ArrayList – contains the agents who are at the street intersection

Actions in the Schedule

step() – runs at every 1 tick.

## 4.4 Other Model Components

A sequence graph is included that tracks the number of times a street node is visited by any agent, the number of times a robbery occurs on the node and the number of times a robbery is deterred by the presence of a cop.

| | |
|---|---|
| ***Name:*** | ***_Tracking*** |
| ***Title:*** | ***Activity Graph*** |

Series:

totalRobberies – cumulative number of robberies at a node
totalDeterred – cumulative number of robberies prevented by a cop at a node
totalIntersect - cumulative number of times more than one agent is at a node

Schedule:

Currently not scheduled.

## 4.5 Random Number Distributions

The use of robust random number generators (RNGs) is essential to producing high quality, scientifically defensible results. This simulation uses the Mersenne Twister RNG for all the random numbers in the model. The same seed is used for each of the random number distributions in a single run of five experiments. The seed is set in the initAgents() action. Two types of random number distributions are used, normal and uniform. The uniform random number distribution is created in the initAgents and used in various actions as documented below. Two normal distribution are created in the initAgents() action for Full Robbery Directed movement model, one as a static variable the other as a regular normal distribution. The static normal distribution is necessary for directed agent movement and is called from the step() action. The Full Street Robbery Random model requires only one normal distribution be created. The following section describes all the random number distributions used in the model, how they are created and in which action they are used.

**Actions That Use RNG'S**

**initCitizens()**

Uniform distributions are used to:

- choose subsets of agents from the totals set for criminal propensity, employment status, number from between 0 and 999.

- select numbers from between (.1,.9) to assign a percent of time left to rec1.

- Represent randomness in an offender's perception of guardianship and suitability of targets.

Normal distributions are used to:

- assign wealth to the agents: Random.createNormal(50,20)

- assign the amount of time to spend at home with a mean of the experimental value and a standard deviation of 10 percent of the mean (very peaked distribution) as is shown below.

```
societyPercHome = 1 - self.SOCIETAL_TIMEAWAY
standardDeviation = (self.MODEL_DAY * societyPercHome)*.10
meanTimeHome = self.MODEL_DAY * societyPercHome
Random.createNormal(meanTimeHome,standardDeviation)
```

- choose a number of nodes to move each turn. A separate random number generator is created and used for this distribution. A global variable is created to hold the distribution so it can be called from the Citizen.step() action (only in the Activity Space model).

```
mtRNG = MersenneTwister(100)
self.NORM_TRAVEL = Normal(6,1,mtRNG)
```

---

**decideRob()**

Uniform distributions are used to:

- represent differences in perceived guardianship, generate number between -2 and 2. This adds or subtracts up to 2 agents to the perceived guardianship value (e.g. a place manager might represent 2 agents, while someone in car might not count for as much).

- represent differences in perceived target suitability, select a number between -1 and 1 which adds to or subtracts from the difference in wealth between offender and target.

- represent the influence of other unknown factors when the decision to offend could go either way based on guardianship and suitability (i.e. the randDecision section of code)

---

**idChangingEmploymentStatus()**

Uniform distribution is used to:

- choose a subset of agents that will change their employment status, generates a series of numbers from between 0 and 999.

**switchActivitySpace()**

Uses two of the same distributions as the initCitizens() action:

Uniform distributions are used to:

- choose subsets of agents from the totals set for criminal propensity, employment status, number from between 0 and 999.
- select numbers from between (.1,.9) to assign a percent of time left to rec1.

Normal distribution (created in initCitizens()) is used to:

- assign the amount of time to spend at home with a mean of the experimental value and a standard deviation of 10 percent of the mean (very peaked distribution) as is shown below.

```
societyPercHome = 1 - self.SOCIETAL_TIMEAWAY
standardDeviation = (self.MODEL_DAY * societyPercHome)*.10
meanTimeHome = self.MODEL_DAY * societyPercHome
Random.createNormal(meanTimeHome,standardDeviation)
```

**initCops()**

Uniform distribution is used to:

- choose place index numbers to which to assign cops. Randomly chooses from the entire set of street nodes, 0 to 16034.

**Citizen.step()**

Normal distribution is used to:

- obtain a random number of nodes to move each turn. The average number of nodes needs to be six per turn. This distribution was created in initAgents() and is called here self.NORM_TRAVEL.

## 4.6 Statistics Files

The following files are written out for each model run:

1) C:/model_output<condition>/modelRunDatav1.csv – in the initCitizens() write out agent values for criminal propensity, employment and wealth; in the initModel() write out parameter values for model
2) C:/model_output<condition>/citizen<agentName>.csv – a series of files (one for each agent) that are written from the writeCitizenTravelInfoFiles(). Each file contains the: tick, the timers for home, main, rec1, rec2, time exposed and time traveling, the position of the agent in their path, number of offenses, number of victimizations and the amount of time originally assigned to spend at each place.
3) C:/model_output<condition>/statistics.csv – writes out the times actually spent at activity nodes. All agents are in one file. Easy to get percentages of time spent at home, main, rec1, rec2 from this file.
4) C:/model_output<condition>/occupiedSnapshot<Tick>.csv – a series of files are created (one at each tick). Documents the distribution of agents across the streetnodes and keeps track of how many agents were at any one street node and which agents were there.
5) C:/model_output<condition>/timeAtActivityNodes.csv – file that has the amounts of time assigned for activity nodes. All agents are in one file.
6) C:/model_output<condition>/citizenChar.csv – prints every 60 ticks. Has information to monitor the model level variables. Files is created by createCitizenTravelOutputFiles() and written to in dataRecorder()
7) C:/model_output<condition>/path<agentName>.csv – creates and writes to the files to hold the list of nodes that each citizen visits during random movement. Action must be scheduled to run.

## *Appendix 3: Street Robbery Model: Simple Version Code*

### Street Robbery Simple Actions

**def initAgents():**

Java imports
uchicago.src.simbuilder.util.MessageDisplay
java.lang.String
anl.repast.gis.data.dbf.DBFReader
anl.repast.gis.data.dbf.JDBField
java.Array
java.util.Vector
java.util.List
java.lang.Object
java.util.ArrayList
uchicago.src.sim.util.Random
java.io.PrintWriter

Code
```
 print "Inside initAgents"
 if (self.messageDisplay == None):
   self.messageDisplay = MessageDisplay()
   self.messageDisplay.display("Street Robbery Messages")
 else:
   self.messageDisplay.clear()

 # Explicitly set the random number generator seed and initialize Random distributions
 # Create RNG and set seed
 Random.setSeed(self.SEED)
 #Random.setSeed(100)
 Random.createUniform()

 # Create log file for model run
 self.writeModelRunData()

 # Initialize model level variables
 self.initModel()

 # Initialize the activity spaces of agents
 self.initCitizensRandom()

 # Create output files for analysis
 self.createCitizenTravelOutputFiles()

 # Process the street nodes for use in the model
 self.setupPlaces()

 # Check to make sure values in shapefile fields are zero
 for node as Place in self.places:
   if node.totalRob > 0 or node.totalVisit > 0 or node.totPrevent > 0:
     print "WARNING:  Shapefile had non-zero values in counter fields"

 # Initialize the cop agents
```

```
  self.initCops()
```

**def updateDisplay():**

```
  #print "Inside updateDisplay"
  self.updateGISDisplay()
```

**def writeAgents():**

```
  #print "Inside writeAgents-Model level"
  baseFilePath = ".\\projects\\rob_model\\shapefiles\\"
  self.writeAgents(self.places, baseFilePath + "strnodes2.shp")
```

**def setupPlaces():**

Java Imports
java.io.BufferedReader
java.io.FileReader
java.util.StringTokenizer

Code

```
  print "Inside setupPlaces"

  # Put Places in a HashMap where the key is the strnode-id
  # Creates the map
  self.placeMap = LinkedHashMap()

  # Add the places to the hashmap
  for currentPlace as Place in self.places:
    specNode = "0"
    specNode = String.valueOf(currentPlace.getSTRCL_())
    specNodeNew = Float(specNode)
    self.placeMap.put(specNodeNew, currentPlace)
    #print "PLACE node info: ", specNodeNew
    currentPlace.setMyNeighbors(ArrayList())

  # Read the neighbors file and set each nodes neighbors.
  # The neighbors files lists the active node and the neighboring
  # nodes of that active node.  The map created above is used to
  # get the neighbors for each active node.

  fileName = "./projects/rob_model/neighborFiles/nodenghbrs.csv"
  reader = BufferedReader(FileReader(fileName))
  line = reader.readLine()

  while(line):
    tokenizer = StringTokenizer(line, ",")
    if(tokenizer.hasMoreTokens()):
      activeNode = tokenizer.nextToken().trim()
      actNodeObject = Float(activeNode)
      currentPlace = (Place)self.placeMap.get(actNodeObject)
      #print "Current variable ", activeNode #prints out the variable strcl_
      #print "Current node from place object:  ", currentPlace.getSTRCL_()
      nghs = currentPlace.getMyNeighbors()
      while (tokenizer.hasMoreTokens()):
        ngh = tokenizer.nextToken()
        currentPlace.myNeighbors.add(ngh)
        #print "Neighbor node ", ngh
  # Read the line
  line = reader.readLine()
 # Close the reader
 reader.close()

 #This code enables verification that the myNeighbors array has the correct values
 for currentPlace as Place in self.places:
  #print "Streetnode: ", node.strcl_
  if currentPlace.getMyNeighbors() == None:
   print "Neighbor arraylist is empty for node " + currentPlace.strcl_
```

**def showMessage(String message):**

Java imports
javax.swing.JOptionPane

Code
```
 print "Inside showMessage"
 JOptionPane.showMessageDialog((JComponent)None, message)
```

**def incrementModel()**:

```
 # Increment the modelStep field
 if self.modelStep < self.MODEL_YEAR:     #525,600
 #if self.modelStep < 40320:      # month is 40,320
  self.modelStep = self.modelStep + 1
 else:
  self.writeFinalAgents()
  for node as Place in self.places:
   node.totalVisit = 0
   node.totalRob = 0
   node.totPrevent = 0
   node.copStart = 0
   node.citiStart = 0
   node.crimStart = 0
  self.writeAgents()
  self.writeStatistics()
  self.dataRecorder()
  print "YEAR OVER"
  self.stop()
 #print "MODEL STEP = ", self.modelStep

 # ActiveNode - call a method to write out a file of the nodes and their
 # associated agents at each step
 #print "Called writeOccupiedNodes"
 #self.writeOccupiedNodes()

 # Write out citizen position
 #self.writeCitizenInfoPaths()

 # Make the decision to commit a crime
 # print "Total Active NOdes: ", self.activeNodes.size()
 self.decideRob()

 # Clear the agents from the activeNodes class
 self.activeNodes.clear()
```

```
  #print "Total active nodes after clear: ", self.activeNodes.size()

  # Increment the timers for agents with criminal propensity
  for citizen as Citizen in self.citizens:
    if citizen.criminalPropensity == true:
      if citizen.timerRepeat > 0 and citizen.timerRepeat < self.REPEAT:
        citizen.timerRepeat = citizen.timerRepeat + 1
        #print "Agent " + citizen.name + " repeat timer incremented to: " + citizen.timerRepeat
      elif citizen.timerRepeat == self.REPEAT:
        #print "REPEAT value: " + self.REPEAT
        citizen.timerRepeat = 0
        #print "Agent " + citizen.name + " timer reset to 0: " + citizen.timerRepeat
      else:
        citizen.timerRepeat = citizen.timerRepeat

  #print "TOTAL Robberies in society: ", self.totRob
```

**def initModel():**

Java imports

```
cern.jet.random.*
cern.jet.random.engine.MersenneTwister
uchicago.src.sim.util.Random
cern.jet.random.Normal
```

Code

```
  print "Inside initModel"

  # Open log file
  logoutput = self.LOG_FILE
  logWriter = BufferedWriter(FileWriter(logoutput, true))

  # Set static field values for model run
  #self.SOCIETAL_TIMEAWAY = .30
  self.modelStep = 0
  self.MODEL_HOUR = 60  #360 steps per hour, Travel occurs at 6 steps per minute
  self.MODEL_DAY = (24 * self.MODEL_HOUR)
  self.MODEL_WEEK = (7 * self.MODEL_DAY)
  self.MODEL_YEAR = (365 * self.MODEL_DAY)
  #self.REPEAT = 60      #Time until a criminal can reoffend

  # Print to file to document model run
  temp =  self.SOCIETAL_TIMEAWAY *100
  logData = "Experimental condition: " + temp + "% time spent away from home" + "::::" + "Number
of Agents in model " + self.AGENTS
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = ""
  logData = "Number of cops: " + self.COPS
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = ""
  logData = "Limit on Repeat Offending: " + self.REPEAT
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = ""


  #Close the log file
  logWriter.close()
```

**def decideRob():**

Java Imports

java.lang.Object
java.lang.String
uchicago.src.sim.util.SimUtilities
java.util.Arrays
java.util.List

Code

```
  #print "Inside decideRob"

  # Open log file
  logoutput = self.LOG_FILE
  logWriter = BufferedWriter(FileWriter(logoutput, true))
  #data = "Inside Decide Rob Action"
  #intSize = int(data.length())
  #logWriter.write(data,0,intSize)
  #logWriter.newLine()
  #data = ""
  # Check each ActiveNode for list of agents
  # Logical error check
  if self.activeNodes.size() > self.AGENTS:
    ###########################
    data = "Too Many Active Nodes during step: " + self.modelStep
    intSize = int(data.length())
    logWriter.write(data,0,intSize)
    logWriter.newLine()
    data = ""

  # Loop through the nodes with citizens and make the decision to commit a robbery
  for occupied as ActiveNode in self.activeNodes:
    # Check agents at each of the active nodes
    #print "The Node being evaluated is: ", occupied.strnode

    #################
    #data = "Street Node " + occupied.strnode
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    # Initialize variables in action
    numAgentsAtNode = occupied.getAgentList().size()
    #data = "Number of Agents at Node (from size of agent list field): " + numAgentsAtNode
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""
    numCrimAtNode = 0
    numAgentAtRisk = 0
    numCriminals = 0
    offenderAtNode = false
    curStreetNode = (Place)self.places.get(0)
    #print "The default curStreetNode: ", curStreetNode.STRCL_
    curAgent = (Citizen)self.citizens.get(0)
    #targetAgent = (Citizen)self.citizens.get(0)
    targetAgent = (Citizen)self.citizens.get(0)
    criminalAgent = (Citizen)self.citizens.get(0)
    copPresent = false
    robbery = true
    crimWealth = 0
    evalWealth = 0
    targetWealth = 0
    suitability = 0
    targetSet = false

    #################
    #data = "START VALUES: Criminal Agent-- " + criminalAgent.name + "Current Agent-- " +
curAgent.name + "Target agent- " + targetAgent.name
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    # Log presence of agents on street node
    # Retrieving the place by converting to a float object
    occupiedObject = Float(occupied.strnode)
    currentPlace = (Place)self.placeMap.get(occupiedObject)
    #print "NEW Place node:", currentPlace.getSTRCL_()
    #print "Number of agents at Node: ", numAgentsAtNode
    #self.messageDisplay.addAlert("There are "+ numAgentsAtNode + " citizens at " +
occupied.strnode)

    # Log fact that agents visited a node in the shapefile
    if (currentPlace != None):
      currentPlace.totalVisit = currentPlace.totalVisit + numAgentsAtNode
      #currentPlace.visits = currentPlace.visits + numAgentsAtNode
      #print "NEW Number of Visits: ", currentPlace.totalVisit
    else:
      print "Unable to log visit at strnode: " + occupied.strnode + " during tick: " + self.modelStep

    # Loop through all the cops to find out if there is a cop at node
    for copAtNode as Cop in self.cops:
      copPlace = copAtNode.getLocation()
      # When you find a cop at the place break out of loop and calculate variable
      if copPlace == currentPlace:
        #print "Cop at node: ", copAtNode.location.STRCL_
        copPresent = true
        break
      else:
        copPresent = false
    ##############################DEBUG
    #if copPresent == true:
      #data = "Cop is at node! "
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
    #if copPresent == true:
      #print "Cop! at node "+ occupied.strnode

    # Only evaluate nodes that have more than one citizen and there is no cop present
    if numAgentsAtNode > 1:

      #################  DEBUG
      #data = "Street Node " + occupied.strnode
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
      ##############################
      #data = "Number of Agents at Node is: " + numAgentsAtNode
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
      #i = 0
      #j = 1

      # Calculate the level of GUARDIANSHIP
      guardianship = (numAgentsAtNode - 2) +
Random.uniform.nextIntFromTo(self.MIN_GUARDIANSHIP,self.MAX_GUARDIANSHIP)
      #print "Guardianship is: ", guardianship

      # Outside loop that checks each of the agents at a particular node using the citizen name
      # Shuffle the agents at a node so they have an equal chance of being selected first and thus
      # are not always evaluated in the same order.

      #for position in range(0, numAgentsAtNode):
        #print "Original Order: Node--" + occupied.strnode + " Position " + position + ", "+
String.valueOf(occupied.getAgentList().get(position))

      # Create a distribution using number of agents at node
      maxValue = numAgentsAtNode-1

      # Create arraylist variables
      # Array to hold randomly shuffled agents
      randList = ArrayList()
      # List of array positions that have been used
      strList = ArrayList()
      foundIt = false
      #print numAgentsAtNode
      # Outside while to create a new list of all the agents at the node in a new order

      while randList.size() < numAgentsAtNode:
        # Generate a random number
        foundIt = false
        index = Random.uniform.nextIntFromTo(0,numAgentsAtNode-1)
        indexStr = String.valueOf(index)
        #print "The first index generated is: " + index
        ##############################
        #data = "Index value: " + index
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""
        # If this is the first agent generated then add it to the new randList array, otherwise check to see if
        # the index has already been used
        if strList.size() >= 1:
         for p in range (0, strList.size()):
          if String.valueOf(strList.get(p)) == indexStr:
           foundIt = true
           break

        if foundIt == false:
         agent = occupied.AgentList.get(index)
         randList.add(agent)
         strList.add(indexStr)
         #print "randList size is ", randList.size()
         #print "New size of list of index numbers is " + strList.size()

      # Code to verify new order
      #for position in range(0, numAgentsAtNode):
        #print "New order at Node: " + occupied.strnode + " position " + position + ", " +
String.valueOf(randList.get(position))


      for i in range (0,numAgentsAtNode):
        #Bunch of code that get the agent name (e.g. a1) and then strips off the first character
        #and pulls the correct Citizen agent using the agent name
        fullName = randList.get(i)
        fullStrName = String.valueOf(fullName)
        partName = fullStrName.substring(1)
        #print "Agent: " + partname + " in agentList"
        # Use the agent's name to find the index number of correct Citizen agent
        index = int(partName) - 1
        curAgent = (Citizen)self.citizens.get(index)
        #print "Citizen in agentList: ", curAgent.name
        #print "Current agent: " + curAgent.name + " is criminal? " + curAgent.criminalPropensity

        #################################
        #data = "Loop through current agents to find Criminal: " + i + "," +
String.valueOf(randList.get(i)) + "," + "criminal: " + curAgent.criminalPropensity + "," + " at risk?: " +
curAgent.atRisk
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""

        # Identifies if any of the agents have criminal propensity and are 'atRisk' and selects the
        # first one it finds to be the active criminal in this interaction
        if curAgent.criminalPropensity == true and curAgent.atRisk == true:
         criminalAgent = curAgent
         if criminalAgent.timerRepeat == 0:
          offenderAtNode = true
```

```
          #else:
            #print "Agent " + criminalAgent.name + " Offender unable to offend yet"
          break        #go directly to next if statement (if offenderAtNode == true:)

      ##################################DEBUG
      #data = "Criminal in interaction: " + criminalAgent.name + "," + "timer: " +
criminalAgent.timerRepeat
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""

      # Loop that uses formulas to evaluate guardianship and target suitability
      if offenderAtNode == true:
        #print "Offender: " + criminalAgent.name + " is at Node and evaluating opportunity"

        # Find out how many civilians are 'at risk' and
        # which 'at risk' civilian at the active node has the most wealth
        for i in range (0,numAgentsAtNode):
          # Get the first agent in the randomly ordered list
          fullName = randList.get(i)
          fullStrName = String.valueOf(fullName)
          partName = fullStrName.substring(1)
          # Use the agent's name to find the index number of correct Citizen agent
          index = int(partName) - 1
          evalAgent = (Citizen)self.citizens.get(index)
          evalWealth = evalAgent.wealth
          crimWealth = criminalAgent.Wealth
          #print "Criminal's Wealth: ", crimWealth
          #print "Evaluated agent: ", evalAgent.name
          #print "Evaluated agent's wealth: ", evalWealth

          # Counter for number of criminals at node
          if evalAgent.criminalPropensity == true:
            numCrimAtNode = numCrimAtNode + 1


          ##############################DEBUG
          #data = "Evaluate Wealth for eval agent: " + evalAgent.name + " has wealth of: " + evalWealth
+ " CrimAgent: " + criminalAgent.name + "has wealth of: " + crimWealth
          #data = "Eval agent: " + "," + evalAgent.name + "," + "Wealth: " + "," + evalWealth + "
CrimAgent: " + criminalAgent.name + "has wealth of: " + crimWealth
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

          # Counter for number of agents at node who are 'at risk' of being robbed (only is counted
          # if there is an offender at the node)
          if (evalAgent.atRisk == true) and (evalAgent.name != criminalAgent.name):
            numAgentAtRisk = numAgentAtRisk + 1

          ##############################
          #data = "Number of agents at risk: " + numAgentAtRisk
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

          # Identify the 'at risk' agent with the most wealth
          if criminalAgent.name != evalAgent.name:
            ##############################
            #data = "Comparing " + criminalAgent.name + " with " + crimWealth + " to evaluated agent: "
+ evalAgent.name + " with " + evalWealth
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            #print "Comparing " + criminalAgent.name + " with " + crimWealth + " to evaluated agent: " +
evalAgent.name + " with " + evalWealth
            if (crimWealth <= evalWealth) and (evalAgent.atRisk == true):
              if evalWealth > targetWealth:
                targetWealth = evalWealth
                #targetAgent = (Citizen)self.citizens.get(index)
                targetAgent = evalAgent
                targetSet = true
                #print "Current Agent with highest wealth ", targetAgent.name

        ######################### DEBUG
        #data = "Identity of selected targetAgent: " + "," + targetAgent.name + "," +
targetAgent.criminalPropensity
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""
        #print "Number of agents at risk", numAgentAtRisk

        ######################Print summary
        #if numCrimAtNode >= 2:
          #data = "Model Step: " + "," + self.modelStep
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""
          #data = "Number Criminals: " + "," + numCrimAtNode + "," + "Number Agents at Risk: " +
numAgentAtRisk
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

        # Decide to Commit Robbery
        # Calculate SUITABILITY of victim but first check to see if there is a targetAgent found
        targetExists = false
        if targetSet == true:
          #print "Current Agent with highest wealth ", targetAgent.name
          suitability = targetWealth - crimWealth +
Random.uniform.nextIntFromTo(self.MIN_SUITABILITY, self.MAX_SUITABILITY)
          #suitability = targetWealth - crimWealth + self.UNI_PERCEPTION.nextInt()
          targetExists = true
        # Series of checks necessary to evaluate guardianship value calculated earlier
        # If G < 1 then there is a lack of capable guardians so commitCrime = true
        # If G = 1 then randomly assign T or F with equal probability
        # If G >= 2 then too many guardians so commitCrime = false
        #print "PreCommit Crime Guardianship is: ", guardianship
        #print "Suitability is: ", suitability

        # Check to make sure a target exists and evaluate suitability and guardianship
        if targetExists == true and suitability >= 0 and guardianship < 1 and copPresent == false:
#commit crime
          # Exchange one units of wealth
          # Subtract one unit from victim
          #print "Victim Name ", targetAgent.name
          #print "Victims current wealth ", targetAgent.wealth
          targetAgent.wealth = targetAgent.wealth - 1
          #print "Victims new wealth ", targetAgent.wealth
          #print "Offender Name: " + criminalAgent.name + "Timer value of: " +
criminalAgent.timerRepeat
          #print "Offenders current wealth ", criminalAgent.wealth

          # Add one unit of wealth to criminal
          criminalAgent.wealth = criminalAgent.wealth + 1
          #print "Offenders new wealth ", criminalAgent.wealth

          # Start the timer until citizen can offend again
          criminalAgent.timerRepeat = 1

          # Code to log the offense for that specific place
          if (currentPlace != None):
            currentPlace.totalRob = currentPlace.totalRob + 1
            #print "Robbery at Node: ", currentPlace.STRCL_
            #print "Total Robberies at Specific Node  ", currentPlace.totalRob
            # Log the offense at model level
            self.totRob = self.totRob + 1

            # Log offending and victimization for agents involved
            criminalAgent.numOffen = criminalAgent.numOffen + 1
            targetAgent.numVict = targetAgent.numVict + 1

            ###################################DEBUG
            #data = "ROBBERY:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

             ###################################DEBUG
            #data = "WEALTH:  CriminalAgent: " + criminalAgent.wealth + "," + "TargetAgent: " +
targetAgent.wealth
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            #if targetAgent.criminalPropensity == true:
                ##################################
                #data = "Model step:  " + self.modelStep
                #intSize = int(data.length())
                #logWriter.write(data,0,intSize)
                #logWriter.newLine()
                #data = ""
                #data = "Target Agent: " + targetAgent.name + "is a criminal"
                #intSize = int(data.length())
                #logWriter.write(data,0,intSize)
                #logWriter.newLine()
                #data = ""

          else:
              print "Unable to log robbery at strnode: " + occupied.strnode + " during tick: " +
self.modelStep

        # Random decision to commit robbery
        elif targetExists == true and suitability >= 0 and guardianship == 1 and copPresent == false:
          randDecision = guardianship + Random.uniform.nextIntFromTo(self.MIN_SUITABILITY,
self.MAX_SUITABILITY)
          #print "Random Decision: ", randDecision
          if randDecision == 1:
            break
          elif randDecision < 1:
            # Exchange one units of wealth
            #print "Random: Victim Name ", targetAgent.name
            #print "Random: Victims current wealth ", targetAgent.wealth
            targetAgent.wealth = targetAgent.wealth - 1
            #print "Victims new wealth ", targetAgent.wealth
            #print "Random: Offender Name " + criminalAgent.name + "Timer value of: " +
criminalAgent.timerRepeat
            #print "Random: Offenders current wealth ", criminalAgent.wealth
            criminalAgent.wealth = criminalAgent.wealth + 1
            #print "Offenders new wealth ", criminalAgent.wealth

            # Start the timer until citizen can offend again
            criminalAgent.timerRepeat = 1

            # Log the offense at the specific place
            if (currentPlace != None):
              currentPlace.totalRob = currentPlace.totalRob + 1
              #print "Robbery at Node: ", currentPlace.STRCL_
              #print "Total Robberies at Specific Node  ", currentPlace.totalRob
              # Log the offense at model level
              self.totRob = self.totRob + 1

              # Log offending and victimization for agents involved
              criminalAgent.numOffen = criminalAgent.numOffen + 1
              targetAgent.numVict = targetAgent.numVict + 1

              ###################################
              #data = "ROBBERY:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
              #intSize = int(data.length())
              #logWriter.write(data,0,intSize)
              #logWriter.newLine()
```

```
             #data = ""

             #if targetAgent.criminalPropensity == true:
               ##################################
               #data = "Target Agent: " + targetAgent.name + "is a criminal"
               #intSize = int(data.length())
               #logWriter.write(data,0,intSize)
               #logWriter.newLine()
               #data = ""

           else:
             print "Unable to log random decision robbery at strnode: " + occupied.strnode + " during tick:
" + self.modelStep

     ##################################
     #data = "FINAL:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
     #intSize = int(data.length())
     #logWriter.write(data,0,intSize)
     #logWriter.newLine()

   # Loop that logs deterrence effect of cops
   #print "Cop was present and I made it inside for loop"
   if offenderAtNode == true and copPresent == true and numAgentAtRisk >= 1 and targetWealth > 0:
     currentPlace.totPrevent = currentPlace.totPrevent + 1
     self.totDeter = self.totDeter + 1
     #print "Running total of crimes DETERRED is: ", self.totDeter

   # Log occurrence of potential crime situation - offender and victims present
   if offenderAtNode == true and numAgentAtRisk >= 1:
     self.totIntersect = self.totIntersect + 1
     #print "Running total of potential robbery situations: ", self.totIntersect

   #else:
     #print "Only one agent at node or cop at node."

  # Close the writer
  logWriter.close()
```

**def writeOccupiedNodes():**

```
  #print "Inside writeOccupiedNodes from model level"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Write out contents of ACTIVE NODES class
  # CREATE an output file and the buffered writer to write the activity times for each agent to a file
  currTick = int(self.getTickCount())
  #print "Current Tick: ", currTick
  outFileName = "C:/model_output"+away+"/occupiedSnapshot"+currTick+".csv"
  #outFileName = "./projects/rob_model/output/occupiedSnapshot"+currTick+".csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames = "StreetNode,NumAgents,Agent1,Agent2,Agent3,Agent4,Agent5"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()
  # Loop through all the active nodes and write the agents at each node to a file
  for occupied as ActiveNode in self.activeNodes:
    #print "Number of agents at node ", occupied.agentList.size()
    #create a string of each data field to be written to the file
    tempNode = String.valueOf(occupied.strnode)
    numAgents = String.valueOf(occupied.agentList.size())
    temp = tempNode + "," + numAgents

    #print numAgents + " are at node"

    for i in range (0,occupied.agentList.size()):
      temp = temp + ","
      temp = temp + String.valueOf(occupied.agentList.get(i))

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

  #print "Agent:", citizens.getName()
  #print "Time Traveling: ", citizens.getTimeTraveling()
  #print "Time at Home: ", citizens.getTimeHome()
  #print "Time at Main: ", citizens.getTimeMain()
  #print "Time at Rec1: ", citizens.getTimeRec1()
  #print "Time at Rec2: ", citizens.getTimeRec2()

  # Close the file of activity path nodes
  txtWriter.close()

def initCops():
  print "Inside init cops"
  # Randomly assign the cops to a starting location.
  # Use the Places to get the strnode
  for i in range (self.COPS):
    #index =  self.UNI_SELPLACES.nextInt()
    index = Random.uniform.nextIntFromTo(0, self.NUM_PLACES - 1)
    #print "Index ", index
    cop = Cop()
    cop.setModel(self)

    node = (Place)self.places.get(index)
    #print "FOUND a place " , node.STRCL_

    # Log that cop started at this node
    node.copStart = 1
    cop.setLocation(node)
    cop.setStrnode(node.STRCL_)
    self.cops.add(cop)

def resetAgentsDaily():
  #print "Inside resetAgentsDaily"

  for citizen as Citizen in self.citizens:
    citizen.atActivity = true
    citizen.atRisk = false
    citizen.moveStatus = false
    citizen.position = 0
    citizen.timeCounter = 0
    citizen.timerHome = 0
    citizen.timerMain = 0
    citizen.timerRec1 = 0
    citizen.timerRec2 = 0
    #print "Counter at reset agent: " + citizen.timerRepeat
```

**def createCitizenTravelOutputFiles():**

```
  print "Inside createCitizenTravelOutputFiles"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Citizen characteristics
  #for citizen as Citizen in self.citizens:
    #agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    #currTick = int(self.getTickCount())
    #outFileName = "C:/model_output"+away+"/citizens"+agentName+".csv"
    #txtWriter = BufferedWriter(FileWriter(outFileName))
    #columnNames =
"Tick,timerHome,timerMain,timerRec1,timerRec2,totTimeTraveling,totTimeExposed,numVict,numOf
fen,assignHome,assignMain,assignRec1,assignRec2,assignTravel"
    #intSize = int(columnNames.length())
    #txtWriter.write(columnNames,0,intSize)
    #txtWriter.newLine()
    #txtWriter.close()

  # Citizen random path nodes
  #for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    #agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    #currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    #outFileName = "C:/model_output"+away+"/path"+agentName+".csv"
    #txtWriter = BufferedWriter(FileWriter(outFileName, true))
    #txtWriter.close()


  # Create an output file for model runtime statistics
  # Average number of agents unemployed, average wealth, robbery rate, exposure rate, etc.
  currTick = int(self.getTickCount())
  outFileName = "C:/model_output"+away+"/citizenChar.csv"
  dataWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Tick,NumChangeEmp,NumUnemployed,TotWealth,AvgWealth,RobRate,TotVictims,RepeatVict,Tot
Offen,RepeatOffen,NumExp,PercExposed,NumTravel,PercTraveling,numActiveOffenders,numWaitin
gOffenders,cumDeter,cumIntersect,cumRobberies"
  intSize = int(columnNames.length())
  dataWriter.write(columnNames,0,intSize)
  dataWriter.newLine()
  dataWriter.close()
```

```
def writeCitizenTravInfotoFiles():
  #print "Inside writeCitizenTravInfotoFiles"
  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Loop through all citizen agents and write out the specified fields
  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    outFileName = "C:/model_output"+away+"/citizens"+agentName+".csv"
    txtWriter = BufferedWriter(FileWriter(outFileName, true))

    tempName = currTick
    home = String.valueOf(citizen.timerHome)
    temp = tempName + "," + home
    main = String.valueOf(citizen.timerMain)
    temp = temp + "," + main
    rec1 = String.valueOf(citizen.timerRec1)
    temp = temp + "," + rec1
    rec2 = String.valueOf(citizen.timerRec2)
    temp = temp + "," + rec2
    travel = String.valueOf(citizen.totTimeTraveling)
    temp = temp + "," + travel
    expose = String.valueOf(citizen.totTimeExposed)
    temp = temp + "," + expose
    vict = String.valueOf(citizen.numVict)
    temp = temp + "," + vict
    offen = String.valueOf(citizen.numOffen)
    temp = temp + "," + offen
    ahome = String.valueOf(citizen.timeHome)
    temp = temp + "," + ahome
    amain = String.valueOf(citizen.timeMain)
    temp = temp + "," + amain
    arec1 = String.valueOf(citizen.timeRec1)
    temp = temp + "," + arec1
    arec2 = String.valueOf(citizen.timeRec2)
    temp = temp + "," + arec2
    atravel = String.valueOf(citizen.timeTraveling)
    temp = temp + "," + atravel

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()
    txtWriter.close()

def writeModelRunData():
  print "Inside writeModelRunData"
  away = int(self.SOCIETAL_TIMEAWAY*100)
  print away

  # CREATE an output file and the buffered writer to write the activity times for each agent to a file
  modelRun = 1
  self.LOG_FILE = "C:/model_output"+away+"/RunDatav"+ modelRun + ".csv"
  txtWriter = BufferedWriter(FileWriter(self.LOG_FILE))

  # Put a model run header
  header =  "Model run: " + modelRun
  intSize = int(header.length())
  txtWriter.write(header,0,intSize)
  txtWriter.newLine()

  # Add parameter information
  seed = Random.getSeed()
  nxtLine = "Random Number Seed: " + seed
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Time to reoffend: " + self.REPEAT
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Wealth: " + "Mean " + self.WEALTH_MEAN + ", Standard Deviation " +
self.WEALTH_SD
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Error Term in Suitability: " + "Minimum " + self.MIN_SUITABILITY + ", Maximum " +
self.MAX_SUITABILITY
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Error Term in Guardianship: " + "Minimum " + self.MIN_GUARDIANSHIP + ",
Maximum " + self.MAX_GUARDIANSHIP
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  # Close text writer
  txtWriter.close()

def writeStatistics():
  # Writes out final statistics for all agents in one file to provide summary statistics
  # Aggregate time spent at home, main, rec1, rec2, travel, and exposed.
  # Assigned time to spend at home, main, rec1, rec2, travel.
  # Total number of offenses and victimizations.
  # Create a file

  away = int(self.SOCIETAL_TIMEAWAY*100)

  outFileName = "C:/model_output"+away+"/statistics.csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Agent,timerHome,timerMain,timerRec1,timerRec2,totTimeTraveling,totTimeExposed,numVict,num
Offen,assignHome,assignMain,assignRec1,assignRec2,assignTravel,criminal,wealth"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()

  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    name = String.valueOf(citizen.name)
    home = String.valueOf(citizen.timerHome)
    temp = name + "," + home
    main = String.valueOf(citizen.timerMain)
    temp = temp + "," + main
    rec1 = String.valueOf(citizen.timerRec1)
    temp = temp + "," + rec1
    rec2 = String.valueOf(citizen.timerRec2)
    temp = temp + "," + rec2
    travel = String.valueOf(citizen.totTimeTraveling)
    temp = temp + "," + travel
    expose = String.valueOf(citizen.totTimeExposed)
    temp = temp + "," + expose
    vict = String.valueOf(citizen.numVict)
    temp = temp + "," + vict
    offen = String.valueOf(citizen.numOffen)
    temp = temp + "," + offen
    ahome = String.valueOf(citizen.timeHome)
    temp = temp + "," + ahome
    amain = String.valueOf(citizen.timeMain)
    temp = temp + "," + amain
    arec1 = String.valueOf(citizen.timeRec1)
    temp = temp + "," + arec1
    arec2 = String.valueOf(citizen.timeRec2)
    temp = temp + "," + arec2
    atravel = String.valueOf(citizen.timeTraveling)
    temp = temp + "," + atravel
    acriminal = String.valueOf(citizen.criminalPropensity)
    temp = temp + "," + acriminal
    awealth = String.valueOf(citizen.wealth)
    temp = temp + "," + awealth

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

  # Close the file
  txtWriter.close()
```

**def dataRecorder():**

```
  #print "DATA RECORDER T0 FILE"

  # Writes out model runtime statistics
  # Average number of agents unemployed, average wealth, robbery rate, exposure rate, etc.

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Open the output file and the buffered writer to write the information to a file
  currTick = int(self.getTickCount())
  outFileName = "C:/model_output"+away+"/citizenChar.csv"
  dataWriter = BufferedWriter(FileWriter(outFileName, true))
```

```
# Count number of agents to change employment status
numChange = 0
#for citizens as Citizen in self.citizens:
  #if citizens.changeEmpStatus == true:
    #numChange = numChange + 1
    #citizens.changeEmpStatus = false

# Count unemployed agents
numUnemployed = 0
numEmployed = 0
#for agent as Citizen in self.citizens:
  #if agent.employmentStatus == false:
    #numUnemployed = numUnemployed + 1
  #elif agent.employmentStatus == true:
    #numEmployed = numEmployed + 1
  #else:
    #print "Employment status not assigned"
#print "Number unemployed is: ", numUnemployed
#print "Number employed is: ", numEmployed

# Calculate average wealth of agents
totWealth = 0
for citizens as Citizen in self.citizens:
  totWealth = totWealth + citizens.wealth
aveWealth = totWealth / self.AGENTS

# Calculate the robbery rate
robRate = self.totRob / self.AGENTS

# Count number of agents victimized
totNumVict = 0
for citizens as Citizen in self.citizens:
  if citizens.numVict > 0:
    totNumVict = totNumVict + 1

# Count number of repeat victims
numRepeatVict = 0
for citizens as Citizen in self.citizens:
  if citizens.numVict > 1:
    numRepeatVict = numRepeatVict + 1

# Count number of offenders
totNumOffenders = 0
for citizens as Citizen in self.citizens:
  if citizens.numOffen > 0:
    totNumOffenders = totNumOffenders + 1

# Count number of repeat offenders
numRepeatOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.numOffen > 1:
    numRepeatOffen = numRepeatOffen + 1

# Calculate the number of citizens at risk of victimization
numExp = 0
```

```
  for citizens as Citizen in self.citizens:
    if citizens.atRisk:
      numExp = numExp + 1
  percExp = ((numExp / self.AGENTS) * 100)

  # Calculate the number of citizens traveling
  numTravel = 0
  for citizens as Citizen in self.citizens:
    if citizens.atActivity == false:
      numTravel = numTravel + 1
  percTravel = ((numTravel / self.AGENTS) * 100)

  # Calculate the number of active offenders (able to offend)
  numActiveOffen = 0
  for citizens as Citizen in self.citizens:
    if citizens.criminalPropensity and citizens.atRisk and citizens.timerRepeat == 0:
      numActiveOffen = numActiveOffen + 1

  # Calculate the number of waiting offenders (not able to offend)
  numWaitingOffen = 0
  for citizens as Citizen in self.citizens:
    if citizens.criminalPropensity and citizens.atRisk and citizens.timerRepeat > 0:
      numWaitingOffen = numWaitingOffen + 1



  # Create a string of each data field to be written to the file
  temp = currTick + "," + numChange + "," + numUnemployed + "," + totWealth
  temp = temp + "," + aveWealth
  temp = temp + "," + robRate
  temp = temp + "," + totNumVict
  temp = temp + "," + numRepeatVict
  temp = temp + "," + totNumOffenders
  temp = temp + "," + numRepeatOffen
  temp = temp + "," + numExp + ","+ percExp
  temp = temp + "," + numTravel + ","+ percTravel
  temp = temp + "," + numActiveOffen
  temp = temp + "," + numWaitingOffen
  temp = temp + "," + self.totDeter
  temp = temp + "," + self.totIntersect
  temp = temp + "," + self.totRob
  intSize = int(temp.length())
  dataWriter.write(temp,0,intSize)
  dataWriter.newLine()

  #Close the file
  dataWriter.close()

def initCitizensRandom():
  print "Inside initCitizensRandom"

  # Randomly assign all citizens to a starting location and name them
  for i in range(1000):
    p = i + 1
    index = Random.uniform.nextIntFromTo(0, self.NUM_PLACES - 1)
    #print "Index ", index
  citizen = Citizen()
  citizen.setModel(self)

  node = (Place)self.places.get(index)
  #print "Assigned to place ", node.STRCL_
  # Log where citizen started random movement
  node.citiStart = 1
  citizen.setLocation(node)
  citizen.setStrnode(node.STRCL_)
  name = "a" + p
  citizen.setName(name)
  self.citizens.add(citizen)

# Open log file
logoutput = self.LOG_FILE
logWriter = BufferedWriter(FileWriter(logoutput, true))
logData =  "Log File Name: " + logoutput
intSize = int(logData.length())
logWriter.write(logData,0,intSize)
logWriter.newLine()

# Randomly assign criminal propensity to 20% of the Citizens
# Assign 200 agents criminal propensity (criminalPropensity = true)
for i in range (200):
 index = Random.uniform.nextIntFromTo(0, self.AGENTS-1)
 agent = (Citizen)self.citizens.get(index)
 #print "Index ", String.valueOf(index)
 #print "Agent Name: "+ agent.getName()+ " is a criminal"
 node = agent.getLocation()

 # Condition to check and make sure citizen was not previously selected to have criminal propensity
 if agent.criminalPropensity == false:
  agent.criminalPropensity = true
  # Log where criminal started random movement
  node.crimStart = 1
 else:
  while agent.criminalPropensity == true:
    index = Random.uniform.nextIntFromTo(0, self.AGENTS-1)
    agent = (Citizen)self.citizens.get(index)
    node = agent.getLocation()
  # Log where criminal started random movement
  node.crimStart = 1
  agent.criminalPropensity = true
 #print "Agent Name: "+ agent.getName()+ " is a criminal"

###Randomly assign time to stay at home to each agent
## Five experimental conditions:
#  30% timeAway = 70% of time at home (1008 minutes)
#  40% timeAway = 60% of time at home
#  50% timeAway = 50% of time at home
#  60% timeAway = 40% of time at home
#  70% timeAway = 30% of time at home
# Create a new random number generator to create a normal distribution
# with a mean of 70 and SD of 10.

# CREATE an output file of times at each node and traveling
# First create the buffered writer to write the activity times for each agent to a file
away = int(self.SOCIETAL_TIMEAWAY*100)
outFileName = "C:/model_output" +away+"/timeAtActivityNodes.csv"
txtWriter = BufferedWriter(FileWriter(outFileName))
columnNames = "Agent,Home"
intSize = int(columnNames.length())
txtWriter.write(columnNames,0,intSize)
txtWriter.newLine()

# Uniform distribution to assign amount of time to spend at home
# Mean is the society mean and SD is ten percent of the mean
societyPercHome = 1 - self.SOCIETAL_TIMEAWAY
standardDeviation = (self.MODEL_DAY * societyPercHome)*.10
meanTimeHome = self.MODEL_DAY * societyPercHome

# Create a normal distribution with specified mean and sd
Random.createNormal(meanTimeHome, standardDeviation)

# Allocate the time to remain at home
for citizens as Citizen in self.citizens:

  # Initialize nodeList array
  citizens.nodeList = ArrayList()

  # Get a new random number for each agent
  timeAtHome = Random.normal.nextInt()
  #print "Random time at home: ", totTimeAtHome

  # While loop to check for size of timeAtHome and reset timeAtHome until
  # it is less than total time in day.
  while timeAtHome > self.MODEL_DAY:
    timeAtHome =  Random.normal.nextInt()

  # Assign variable value to field in citizen
  citizens.timeHome = timeAtHome

  # Create a string of each data field to be written to the file
  tempName = String.valueOf(citizens.getName())
  tempHome = String.valueOf(citizens.getTimeHome())
  values = (tempName + "," + tempHome)
  intSize = int(values.length())
  txtWriter.write(values,0,intSize)
  txtWriter.newLine()

  #print "Agent:", citizens.getName()
  #print "Time at Home: ", citizens.getTimeHome()

# Close the file of time at home
txtWriter.close()

# Create new random normal distribution to ASSIGN WEALTH
Random.createNormal(self.WEALTH_MEAN,self.WEALTH_SD)

# ASSIGN wealth to agents
for citizens as Citizen in self.citizens:
  #Get a new random number for each agent
```

```
    citizens.wealth = Random.normal.nextInt()
    #print "Name: ", citizens.getName() + " has been assigned Wealth of: "+ citizens.getWealth()

  # Write out initial values for each agent
  # Header line
  logData = "Name, Criminality, Wealth, TimeHome, StartNode"
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = ""
  for citizens as Citizen in self.citizens:
    # Write values
    logData = citizens.getName() + "," + citizens.getCriminalPropensity() + "," + citizens.getWealth() +
"," + citizens.getTimeHome() + "," + citizens.getStrnode()
    intSize = int(logData.length())
    logWriter.write(logData,0,intSize)
    logWriter.newLine()
    logData = ""

 #Close the log file
  logWriter.close()
```

**def writeFinalAgents():**

```
  print "Writing Final Agents"
  away = int(self.SOCIETAL_TIMEAWAY*100)
  baseFilePath = "C:/model_output"+away+"/"
  self.writeAgents(self.places, baseFilePath + "strnodes"+away+".shp")
```

**def writeCitizenInfoPaths():**

```
  #print "Inside writeCitizenTravInfotoFiles"
  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Loop through all citizen agents and write out the specified fields
  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    outFileName = "C:/model_output"+away+"/path"+agentName+".csv"
    txtWriter = BufferedWriter(FileWriter(outFileName, true))
    node = citizen.strnode
    temp = node + "," + currTick
    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

    txtWriter.close()
```

**Place Actions**
(none)

**Citizen Actions**

**def step():**
<u>Java imports</u>
java.lang.Object
java.lang.Double
java.lang.Number
java.lang.Integer

<u>Code</u>
```
 #print "INSIDE CITIZEN STEP"

 # Every citizen agent evaluates their move status, if they are moving they are added to the active
nodes
 # class and are part of the decision to commit a crime.  Then the values (atRisk, atActivity, moving,
and
 # position are set for the next turn.
 isActNodePosition = 0
 isActNode = 0

 # Each agent stays at home a designated amount of time and then moves with each model tick
 # Increment timerHome value and set moveStatus
 if self.timerHome < self.timeHome:
   self.timerHome = self.timerHome + 1

   # Add the new node to the agent's nodeList field to track
   #self.nodeList.add(String.valueOf(self.strnode))
 else:
   self.moveStatus = true
   self.atRisk = true
   self.totTimeExposed = self.totTimeExposed + 1
   places = self.model.getPlaces()

   #if self.name == "a6" or  self.name == "a7":
    #print "Agent: ", self.name
    #print "Old node: ", self.strnode

   # Identify number of neighbor nodes, generate a random number and use that to pick the next node
   numNeighs = self.location.myNeighbors.size()
   #print "Number of neighbors is: ", numNeighs

   # Generate a value
   index = Random.uniform.nextIntFromTo(0,numNeighs-1)
   #print "Index picked: ",index

   #for node in range (0, numNeighs):
     #print "Neighbor " + node + " is :" + String.valueOf(self.location.getMyNeighbors().get(node))+ "
at position " + index

   # Get the new node and assign it to strnode field
   newNode = self.location.getMyNeighbors().get(index)
   self.strnode = int(String.valueOf(newNode))

   # Add the new node to the agent's nodeList field to track
   #self.nodeList.add(newNode)
```

```
    #print self.name + " Move to " + "adjacent strnode: ", self.strnode
    #print self.name + " Move to " + "adjacent strnode: ", self.nodeList.get(0)

    # Do the assignment directly of the strnode to a place
    nodeFL = Float(self.strnode)
    newLocation = (Place)self.model.placeMap.get(nodeFL)
    self.location = newLocation
    #if self.name == "a6" or  self.name == "a7":
      #print "New location: ", self.location.STRCL_

    # ADD an agent to the ActiveNode class.  If there is an ActiveNode agent
    # that exists with a particular strnode value then add the name of the
    # citizen agent to the agentList (an arrayList).  If there is no ActiveNode
    # with the same value as the currentNode then add a new ActiveNode agent and
    # populate the strnode number with the currentNode and add the name of the
    # citizen agent to the agentList (an arrayList).

    #print "There are " + self.model.activeNodes.size() + "active nodes"

    # Test to see if this is the first ActiveNode
    nodeisEqual = false
    if self.model.activeNodes.size() <> 0:
      for occupied as ActiveNode in self.model.activeNodes:
        if self.strnode == occupied.strnode:
          occupied.agentList.add(self.name)
          nodeisEqual = true
          #print "Found an existing active node"

    if self.model.activeNodes.size() == 0 or nodeisEqual == false:
      newAgent = ActiveNode()
      newAgent.setModel(self.model)
      newAgent.strnode = self.strnode
      newAgent.agentList = ArrayList()
      newAgent.agentList.add(self.name)
      self.model.activeNodes.add(newAgent)
      #print "First agent at node: " + newAgent.strnode
      #print "ArrayList Value = ", newAgent.agentList.get(0)

    #print "Current Node as Integer: ", Integer.toString(self.location)
```

**def payCitizens():**

```
    #print "Inside Pay Citizens"
  # Each employed citizen gets paid at designated intervals

  if self.employmentStatus == true:
    #print "Agent Name: ",self.name
    #print "Agent Old Wealth: ", self.wealth

    self.wealth = self.wealth + 5

    #print "Agent New Wealth: ", self.wealth
```

**Active Node Actions**
(none)

**Cop Actions**

**def step():**
<u>Java imports</u>
java.lang.Object
java.lang.String
uchicago.src.sim.util.SimUtilities
java.util.Arrays
java.util.List

<u>Code</u>

```
  # Every cop moves with each model tick
  places = self.model.getPlaces()
  #print "Old node: ", self.strnode

  # Shuffle the adjacent nodes of the Place where the cop is located
  # Identify number of neighbor nodes
  numNeighs = self.location.myNeighbors.size()
  maxValue = numNeighs-1

  # Generate a value
  index = Random.uniform.nextIntFromTo(0,numNeighs -1)

  #print "Move to index: " + index

  # Verification code
  #for node in range (0, numNeighs):
    #print "Neighbor " + node + " is :" + String.valueOf(self.location.getMyNeighbors().get(node))

  # Get the new node and assign it to strnode field
  # (can't just use index because index and strnode do not correspond)
  newNode = self.location.getMyNeighbors().get(index)
  self.strnode = int(String.valueOf(newNode))

  #print "New strnode: ", self.strnode

  # Do the assignment directly of the strnode to a place
  nodeFL = Float(self.strnode)
  newLocation = (Place)self.model.placeMap.get(nodeFL)
  self.location = newLocation
  #print "New location: ", self.location.STRCL_
```

***Sequence Graph***

```
totalRobberies
return self.totRob

totalDeterred
return self.totDeter

totalIntersect
return self.totIntersect
```

## *Appendix 4: Street Robbery Model: Temporal Version Code*

### ***Street Robbery Temporal Actions***

**def initAgents():**

Java imports

```
uchicago.src.simbuilder.util.MessageDisplay
java.lang.String
anl.repast.gis.data.dbf.DBFReader
anl.repast.gis.data.dbf.JDBField
java.Array
java.util.Vector
java.util.List
java.lang.Object
java.util.ArrayList
uchicago.src.sim.util.Random
java.io.PrintWriter
```

Code

```
  print "Inside initAgents"
  if (self.messageDisplay == None):
    self.messageDisplay = MessageDisplay()
    self.messageDisplay.display("Street Robbery Messages")
  else:
    self.messageDisplay.clear()

  # Explicitly set the random number generator seed and initialize Random distributions
  # Create RNG and set seed
  Random.setSeed(100)
  Random.createUniform()

  # Create log file for model run
  self.writeModelRunData()

  # Initialize model level variables
  self.initModel()

  # Process the street nodes for use in the model
  self.setupPlaces()

  # Initialize the agents
  self.initCitizensRandom()

  # Create output files for analysis
  self.createCitizenTravelOutputFiles()

  # Check to make sure values in shapefile fields are zero
  for node as Place in self.places:
    if node.totalRob > 0 or node.totalVisit > 0 or node.totPrevent > 0:
      print "WARNING:  Shapefile had non-zero values in counter fields"

  # Initialize the cop agents
  self.initCops()
```

```
 print "Leaving initAgents"
```

**def updateDisplay():**

```
 #print "Inside updateDisplay"
 self.updateGISDisplay()
```

**def writeAgents():**

```
 #print "Inside writeAgents-Model level"
 baseFilePath = ".\\projects\\rob_model\\shapefiles\\"
 self.writeAgents(self.places, baseFilePath + "strnodes2.shp")
```

**def setupPlaces():**

<u>Java imports</u>

```
ava.io.BufferedReader
java.io.FileReader
java.util.StringTokenizer
```

<u>Code</u>

```
 print "Inside setupPlaces"

 # Put Places in a HashMap where the key is the strnode-id
 # Creates the map
 self.placeMap = LinkedHashMap()

 # Add the places to the hashmap
 for currentPlace as Place in self.places:
   specNode = "0"
   specNode = String.valueOf(currentPlace.getSTRCL_())
   specNodeNew = Float(specNode)
   self.placeMap.put(specNodeNew, currentPlace)
   #print "PLACE node info: ", specNodeNew
   currentPlace.setMyNeighbors(ArrayList())

 # Read the neighbors file and set each nodes neighbors.
 # The neighbors files lists the active node and the neighboring
 # nodes of that active node.  The map created above is used to
 # get the neighbors for each active node.

 fileName = "./projects/rob_model/neighborFiles/nodenghbrs.csv"
 reader = BufferedReader(FileReader(fileName))
 line = reader.readLine()

 while(line):
   tokenizer = StringTokenizer(line, ",")
   if(tokenizer.hasMoreTokens()):
     activeNode = tokenizer.nextToken().trim()
     actNodeObject = Float(activeNode)
     currentPlace = (Place)self.placeMap.get(actNodeObject)
     #print "Current variable ", activeNode #prints out the variable strcl_
     #print "Current node from place object:  ", currentPlace.getSTRCL_()
     nghs = currentPlace.getMyNeighbors()
     while (tokenizer.hasMoreTokens()):
       ngh = tokenizer.nextToken()
       currentPlace.myNeighbors.add(ngh)
       #print "Neighbor node ", ngh
   # Read the line
```

```
    line = reader.readLine()
  # Close the reader
  reader.close()

  #This code enables verification that the myNeighbors array has the correct values
  #for currentPlace as Place in self.places:
    #print "Streetnode: ", node.strcl_
    #if currentPlace.getMyNeighbors() == None:
      #print "Neighbor arraylist is empty for node " + currentPlace.strcl_
    #else:
      #print "Neighbor Nodes", currentPlace.getMyNeighbors().size()
```

**def showMessage(String message):**

Java imports

javax.swing.JOptionPane

Code

```
  print "Inside showMessage"
  JOptionPane.showMessageDialog((JComponent)None, message)
```

**def incrementModel():**

```
  #print "Inside incrementModel"

   # Increment the modelStep field
  #if self.modelStep < self.MODEL_DAY:     #1,440
  #if self.modelStep < self.MODEL_WEEK:
  if self.modelStep <  self.MODEL_YEAR:       #525,600
  #if self.modelStep < 40320:      # month is 40,320
    self.modelStep = self.modelStep + 1
  else:
    self.writeFinalAgents()
    for node as Place in self.places:
      node.totalVisit = 0
      node.totalRob = 0
      node.totPrevent = 0
      node.copStart = 0
      node.citiStart = 0
      node.crimStart = 0
    self.writeAgents()
    self.writeStatistics()
    self.dataRecorder()
    print "YEAR OVER"
    self.stop()
  #print "MODEL STEP = ", self.modelStep

  # ActiveNode - call a method to write out a file of the nodes and their
  # associated agents at each step
  #print "Called writeOccupiedNodes"
  #self.writeOccupiedNodes()

  # Write out citizen position
  #self.writeCitizenInfoPaths()

  # Make the decision to commit a crime
```

```
 # print "Total Active Nodes: ", self.activeNodes.size()
 self.decideRob()

 # Clear the agents from the activeNodes class
 self.activeNodes.clear()
 #print "Total active nodes after clear: ", self.activeNodes.size()

  # Increment the timers for agents with criminal propensity
 for citizen as Citizen in self.citizens:
   if citizen.criminalPropensity == true:
     if citizen.timerRepeat > 0 and citizen.timerRepeat < self.REPEAT:
       citizen.timerRepeat = citizen.timerRepeat + 1
       #print "Agent " + citizen.name + " repeat timer incremented to: " + citizen.timerRepeat
     elif citizen.timerRepeat == self.REPEAT:
       #print "REPEAT value: " + self.REPEAT
       citizen.timerRepeat = 0
       #print "Agent " + citizen.name + " timer reset to 0: " + citizen.timerRepeat
     else:
       citizen.timerRepeat = citizen.timerRepeat

 #print "TOTAL Robberies in society: ", self.totRob
```

**def initModel():**

Java Imports

```
cern.jet.random.*
cern.jet.random.engine.MersenneTwister
uchicago.src.sim.util.Random
cern.jet.random.Normal
```

Code

```
 print "Inside initModel"

 # Open log file
 logoutput = self.LOG_FILE
 logWriter = BufferedWriter(FileWriter(logoutput, true))

 # Set static field values for model run
 self.modelStep = 0
 self.MODEL_HOUR = 60  #360 steps per hour, Travel occurs at 6 steps per minute
 self.MODEL_DAY = (24 * self.MODEL_HOUR)
 self.MODEL_WEEK = (7 * self.MODEL_DAY)
 self.MODEL_YEAR = (365 * self.MODEL_DAY)

# Print to file to document model run
 temp =  self.SOCIETAL_TIMEAWAY *100
 logData = "Experimental condition: " + temp + "% time spent away from home" + "::::" + "Number
of Agents in model " + self.AGENTS
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
 logData = ""
 logData = "Number of cops: " + self.COPS
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
```

```
  logData = ""

  #Close the log file
  logWriter.close()
```

**def decideRob():**
Java imports
java.lang.Object
java.lang.String
uchicago.src.sim.util.SimUtilities
java.util.Arrays
java.util.List

Code
```
  #print "Inside decideRob"

  # Open log file
  logoutput = self.LOG_FILE
  logWriter = BufferedWriter(FileWriter(logoutput, true))
  #data = "Inside Decide Rob Action"
  #intSize = int(data.length())
  #logWriter.write(data,0,intSize)
  #logWriter.newLine()
  #data = ""

  # Check each ActiveNode for list of agents
  # Logical error check
  if self.activeNodes.size() > self.AGENTS:
    data = "Too Many Active Nodes during step: " + self.modelStep
    intSize = int(data.length())
    logWriter.write(data,0,intSize)
    logWriter.newLine()
    data = ""

  # Loop through the nodes with citizens and make the decision to commit a robbery
  for occupied as ActiveNode in self.activeNodes:
    # Check agents at each of the active nodes
    #print "The Node being evaluated is: ", occupied.strnode

    #################
    #data = "Street Node " + occupied.strnode
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    # Initialize variables in action
    numAgentsAtNode = occupied.getAgentList().size()
    #data = "Number of Agents at Node (from size of agent list field): " + numAgentsAtNode
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    numCrimAtNode = 0
    numAgentAtRisk = 0
    numCriminals = 0
    offenderAtNode = false
    curStreetNode = (Place)self.places.get(0)
    #print "The default curStreetNode: ", curStreetNode.STRCL_
    curAgent = (Citizen)self.citizens.get(0)
    targetAgent = (Citizen)self.citizens.get(0)    #used to initialize target variable
    criminalAgent = (Citizen)self.citizens.get(0)
    copPresent = false
    robbery = true
    crimWealth = 0
    evalWealth = 0
    targetWealth = 0
    suitability = 0
    targetSet = false

    #################
    #data = "START VALUES: Criminal Agent-- " + criminalAgent.name + "Current Agent-- " +
  curAgent.name + "Target agent- " + targetAgent.name
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    # Log presence of agents on street node
    # Retrieving the place by converting to a float object
    occupiedObject = Float(occupied.strnode)
    currentPlace = (Place)self.placeMap.get(occupiedObject)
    #print "NEW Place node:", currentPlace.getSTRCL_()
    #print "Number of agents at Node: ", numAgentsAtNode
    #self.messageDisplay.addAlert("There are "+ numAgentsAtNode + " citizens at " +
  occupied.strnode)

    # Log fact that agents visited a node in the shapefile
    if (currentPlace != None):
      currentPlace.totalVisit = currentPlace.totalVisit + numAgentsAtNode
      #currentPlace.visits = currentPlace.visits + numAgentsAtNode
      #print "NEW Number of Visits: ", currentPlace.totalVisit
    else:
      print "Unable to log visit at strnode: " + occupied.strnode + " during tick: " + self.modelStep

    #Loop through all the cops to find out if there is a cop at node
    for copAtNode as Cop in self.cops:
      copPlace = copAtNode.getLocation()
      #When you find a cop at the place break out of loop and calculate variable
      if copPlace == currentPlace:
        #print "Cop at node: ", copAtNode.location.STRCL_
        copPresent = true
        break
      else:
        copPresent = false

    ##############################DEBUG
    #if copPresent == true:
      #data = "Cop is at node! "
      #intSize = int(data.length())
```

```
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
    #if copPresent == true:
      #print "Cop! at node "+ occupied.strnode

    #Only evaluate nodes that have more than one citizen and there is no cop present
    if numAgentsAtNode > 1:                #Change to > 1 for final testing

      #################  DEBUG
      #data = "Street Node " + occupied.strnode
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
      ############################
      #data = "Number of Agents at Node is: " + numAgentsAtNode
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""

      # Calculate the level of GUARDIANSHIP
      guardianship = (numAgentsAtNode - 2) +
Random.uniform.nextIntFromTo(self.MIN_GUARDIANSHIP,self.MAX_GUARDIANSHIP)
      #print "Guardianship is: ", guardianship

      # Outside loop that checks each of the agents at a particular node using the citizen name
      # Use code to randomly shuffle the agents at a node so they have an equal chance of being
      # selected first and thus are not always evaluated in the same order.

      #for position in range(0, numAgentsAtNode):
        #print "Original Order: Node--" + occupied.strnode + " Position " + position + ", "+
String.valueOf(occupied.getAgentList().get(position))

      # Create a distribution using number of agents at node
      maxValue = numAgentsAtNode-1

      # Create arraylist variables
      # Array to hold randomly shuffled agents
      randList = ArrayList()
      # List of array positions that have been used
      strList = ArrayList()
      foundIt = false
      #print numAgentsAtNode
      # Outside while to create a new list of all the agents at the node in a new order

      while randList.size() < numAgentsAtNode:
        # Generate a random number
        foundIt = false
        #index = shuffleDist.nextInt()
        index = Random.uniform.nextIntFromTo(0,numAgentsAtNode-1)
        indexStr = String.valueOf(index)
        #print "Original index generated is: " + index

        ############################
        #data = "Index value: " + index
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logwriter.newLine()
        #data = ""

        # If this is the first agent generated then add it to the new randList array, otherwise check to see if
        # the index has already been used
        if strList.size() >= 1:
          for p in range (0, strList.size()):
            if String.valueOf(strList.get(p)) == indexStr:
              foundIt = true
              break

        if foundIt == false:
          agent = occupied.AgentList.get(index)
          randList.add(agent)
          strList.add(indexStr)
          #print "randList size is ", randList.size()
          #print "New size of list of index numbers is " + strList.size()

      # Code to verify new order
      #for position in range(0, numAgentsAtNode):
        #print "New order at Node: " + occupied.strnode + " position " + position + ", " +
String.valueOf(randList.get(position))

      for i in range (0,numAgentsAtNode):
        #Bunch of code that get the agent name (e.g. a1) and then strips off the first character
        #and pulls the correct Citizen agent using the agent name
        fullName = randList.get(i)
        fullStrName = String.valueOf(fullName)
        partName = fullStrName.substring(1)
        #print "Agent: " + partname + " in agentList"
        # Use the agent's name to find the index number of correct Citizen agent
        index = int(partName) - 1
        curAgent = (Citizen)self.citizens.get(index)
        #print "Citizen in agentList: ", curAgent.name
        #print "Current agent: " + curAgent.name + " is criminal? " + curAgent.criminalPropensity

        ##################################
        #data = "Loop through current agents to find Criminal: " + i + "," +
String.valueOf(randList.get(i)) + "," + "criminal: " + curAgent.criminalPropensity + "," + " at risk?: " +
curAgent.atRisk
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""

        # Identifies if any of the agents have criminal propensity and are 'atRisk' and selects the
        # first one it finds to be the active criminal in this interaction
        if curAgent.criminalPropensity == true and curAgent.atRisk == true:
          criminalAgent = curAgent
          if criminalAgent.timerRepeat == 0:
            offenderAtNode = true
          #else:
            #print "Agent " + criminalAgent.name + " Offender unable to offend yet"
        break        #go directly to next if statement (if offenderAtNode == true:)

      ##################################DEBUG
      #data = "Criminal in interaction: " + criminalAgent.name + "," + "timer: " +
criminalAgent.timerRepeat
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""


      # Loop that uses formulas to evaluate guardianship and target suitability
      if offenderAtNode == true:
        #print "Offender: " + criminalAgent.name + " is at Node and evaluating opportunity"

        # Find out how many civilians are 'at risk' and
        # which 'at risk' civilian at the active node has the most wealth
        for i in range (0,numAgentsAtNode):
          # Get the first agent in the randomly ordered list
          fullName = randList.get(i)
          fullStrName = String.valueOf(fullName)
          partName = fullStrName.substring(1)
          # Use the agent's name to find the index number of correct Citizen agent
          index = int(partName) - 1
          evalAgent = (Citizen)self.citizens.get(index)
          evalWealth = evalAgent.wealth
          crimWealth = criminalAgent.Wealth
          #print "Criminal's Wealth: ", crimWealth
          #print "Evaluated agent: ", evalAgent.name
          #print "Evaluated agent's wealth: ", evalWealth

          # Counter for number of criminals at node
          if evalAgent.criminalPropensity == true:
            numCrimAtNode = numCrimAtNode + 1


          ##############################DEBUG
          #data = "Evaluate Wealth for eval agent: " + evalAgent.name + " has wealth of: " + evalWealth
+ " CrimAgent: " + criminalAgent.name + "has wealth of: " + crimWealth
          #data = "Eval agent: " + "," + evalAgent.name + "," + "Wealth: " + "," + evalWealth + "
CrimAgent: " + criminalAgent.name + "has wealth of: " + crimWealth
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

          # Counter for number of agents at node who are 'at risk' of being robbed (only is counted
          # if there is an offender at the node)
          if (evalAgent.atRisk == true) and (evalAgent.name != criminalAgent.name):
            numAgentAtRisk = numAgentAtRisk + 1

          ##############################
          #data = "Number of agents at risk: " + numAgentAtRisk
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

          # Identify the 'at risk' agent with the most wealth
          if criminalAgent.name != evalAgent.name:
            #############################
            #data = "Comparing " + criminalAgent.name + " with " + crimWealth + " to evaluated agent: "
+ evalAgent.name + " with " + evalWealth
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            #print "Comparing " + criminalAgent.name + " with " + crimWealth + " to evaluated agent: " +
evalAgent.name + " with " + evalWealth
            if (crimWealth <= evalWealth) and (evalAgent.atRisk == true):
              if evalWealth > targetWealth:
                targetWealth = evalWealth
                #targetAgent = (Citizen)self.citizens.get(index)
                targetAgent = evalAgent
                targetSet = true
                #print "Current Agent with highest wealth ," + targetAgent.name

        ######################## DEBUG
        #data = "Identity of selected targetAgent: " + "," + targetAgent.name + "," +
targetAgent.criminalPropensity
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""
        #print "Number of agents at risk", numAgentAtRisk

        ######################Print summary
        #if numCrimAtNode >= 2:
          #self.multiCriminalsAtNode = self.multiCriminalsAtNode + 1
          #data = "Two or more criminals presentat node during Model Step: " + "," + self.modelStep
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""
          #data = "Number Criminals: " + "," + numCrimAtNode + "," + "Number Agents at Risk: " +
numAgentAtRisk
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

        # Decide to Commit Robbery
        # Calculate SUITABILITY of victim but first check to see if there is a targetAgent found
        targetExists = false
        if targetSet == true:
          #print "Current Agent with highest wealth ", targetAgent.name
          suitability = targetWealth - crimWealth +
Random.uniform.nextIntFromTo(self.MIN_SUITABILITY, self.MAX_SUITABILITY)
          targetExists = true

        # Series of checks necessary to evaluate guardianship value calculated earlier
        # If G < 1 then there is a lack of capable guardians so commitCrime = true
        # If G = 1 then randomly assign T or F with equal probability
        # If G >= 2 then too many guardians so commitCrime = false
        #print "PreCommit Crime Guardianship is: ", guardianship
        #print "Suitability is: ", suitability

        # Check to make sure a target exists and evaluate suitability and guardianship
        if targetExists == true and suitability >= 0 and guardianship < 1 and copPresent == false:
#commit crime
          # Exchange one units of wealth
          # Subtract one unit from victim
          #print "Victim Name ", targetAgent.name
          #print "Victims current wealth ", targetAgent.wealth
          targetAgent.wealth = targetAgent.wealth - 1
          #print "Victims new wealth ", targetAgent.wealth
          #print "Offender Name: " + criminalAgent.name + "Timer value of: " +
criminalAgent.timerRepeat
          #print "Offenders current wealth ", criminalAgent.wealth

          # Add one unit of wealth to criminal
          criminalAgent.wealth = criminalAgent.wealth + 1
          #print "Offenders new wealth ", criminalAgent.wealth

          # Start the timer until citizen can offend again
          criminalAgent.timerRepeat = 1

          # Code to log the offense for that specific place
          if (currentPlace != None):
            currentPlace.totalRob = currentPlace.totalRob + 1
            #print "Robbery at Node: ", currentPlace.STRCL_
            #print "Total Robberies at Specific Node  ", currentPlace.totalRob
            # Log the offense at model level
            self.totRob = self.totRob + 1

            # Log offending and victimization for agents involved
            criminalAgent.numOffen = criminalAgent.numOffen + 1
            targetAgent.numVict = targetAgent.numVict + 1

            ###################################DEBUG
            #data = "ROBBERY:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            ###################################DEBUG
            #data = "WEALTH:  CriminalAgent: " + criminalAgent.wealth + "," + "TargetAgent: " +
targetAgent.wealth
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            if targetAgent.criminalPropensity == true:
                ###################################
```

```
                data = "Model step:  " + self.modelStep
                intSize = int(data.length())
                logWriter.write(data,0,intSize)
                logWriter.newLine()
                data = ""
                data = "Target Agent: " + targetAgent.name + "is a criminal"
                intSize = int(data.length())
                logWriter.write(data,0,intSize)
                logWriter.newLine()
                data = ""

          else:
              print "Unable to log robbery at strnode: " + occupied.strnode + " during tick: " +
self.modelStep

        # Random decision to commit robbery
        elif targetExists == true and suitability >= 0 and guardianship == 1 and copPresent == false:
          randDecision = guardianship + Random.uniform.nextIntFromTo(self.MIN_SUITABILITY,
self.MAX_SUITABILITY)
          #print "Random Decision: ", randDecision
          if randDecision == 1:
            break
          elif randDecision < 1:
            # Exchange one units of wealth
            #print "Random: Victim Name ", targetAgent.name
            #print "Random: Victims current wealth ", targetAgent.wealth
            targetAgent.wealth = targetAgent.wealth - 1
            #print "Victims new wealth ", targetAgent.wealth
            #print "Random: Offender Name " + criminalAgent.name + "Timer value of: " +
criminalAgent.timerRepeat
            #print "Random: Offenders current wealth ", criminalAgent.wealth
            criminalAgent.wealth = criminalAgent.wealth + 1
            #print "Offenders new wealth ", criminalAgent.wealth

            # Start the timer until citizen can offend again
            criminalAgent.timerRepeat = 1

            # Log the offense at the specific place
            if (currentPlace != None):
              currentPlace.totalRob = currentPlace.totalRob + 1
              #print "Robbery at Node: ", currentPlace.STRCL_
              #print "Total Robberies at Specific Node  ", currentPlace.totalRob
              # Log the offense at model level
              self.totRob = self.totRob + 1

              # Log offending and victimization for agents involved
              criminalAgent.numOffen = criminalAgent.numOffen + 1
              targetAgent.numVict = targetAgent.numVict + 1

              ##################################
              #data = "ROBBERY:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
              #intSize = int(data.length())
              #logWriter.write(data,0,intSize)
              #logWriter.newLine()
              #data = ""
              #if targetAgent.criminalPropensity == true:
                ##################################
                #data = "Target Agent: " + targetAgent.name + "is a criminal"
                #intSize = int(data.length())
                #logWriter.write(data,0,intSize)
                #logWriter.newLine()
                #data = ""

            else:
              print "Unable to log random decision robbery at strnode: " + occupied.strnode + " during tick:
" + self.modelStep

      ##################################
      #data = "FINAL:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()

    # Loop that logs deterrence effect of cops
    #print "Cop was present and I made it inside for loop"
    if offenderAtNode == true and copPresent == true and numAgentAtRisk >= 1 and targetWealth > 0:
      currentPlace.totPrevent = currentPlace.totPrevent + 1
      self.totDeter = self.totDeter + 1
      #print "Running total of crimes DETERRED is: ", self.totDeter

    # Log occurrence of potential crime situation - offender and victims present
    if offenderAtNode == true and numAgentAtRisk >= 1:
      self.totIntersect = self.totIntersect + 1
      #print "Running total of potential robbery situations: ", self.totIntersect

    #else:
      #print "Only one agent at node or cop at node."

  # Close the writer
  logWriter.close()
```

**def writeOccupiedNodes():**

```
  #print "Inside writeOccupiedNodes from model level"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Write out contents of ACTIVE NODES class
  # CREATE an output file and the buffered writer to write the activity times for each agent to a file
  currTick = int(self.getTickCount())
  #print "Current Tick: ", currTick
  outFileName = "C:/model_output"+away+"/occupiedSnapshot"+currTick+".csv"
  #outFileName = "./projects/rob_model/output/occupiedSnapshot"+currTick+".csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames = "StreetNode,NumAgents,Agent1,Agent2,Agent3,Agent4,Agent5"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()
  # Loop through all the active nodes and write the agents at each node to a file
  for occupied as ActiveNode in self.activeNodes:
    #print "Number of agents at node ", occupied.agentList.size()
    #create a string of each data field to be written to the file
    tempNode = String.valueOf(occupied.strnode)
    numAgents = String.valueOf(occupied.agentList.size())
    temp = tempNode + "," + numAgents

    for i in range (0,occupied.agentList.size()):
      temp = temp + ","
      temp = temp + String.valueOf(occupied.agentList.get(i))

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

  #print "Agent:", citizens.getName()
  #print "Time Traveling: ", citizens.getTimeTraveling()
  #print "Time at Home: ", citizens.getTimeHome()
  #print "Time at Main: ", citizens.getTimeMain()
  #print "Time at Rec1: ", citizens.getTimeRec1()
  #print "Time at Rec2: ", citizens.getTimeRec2()

  # Close the file of activity path nodes
  txtWriter.close()


def idChangingEmploymentStatus():
  print "ID CHANGE AGENTS loop"

  #SWITCH the employment status for 3% of agents in the model using a uniform distribution
  for i in range (30):
   index = Random.uniform.nextIntFromTo(0,self.AGENTS - 1)
   agent = (Citizen)self.citizens.get(index)
   agent.changeEmpStatus = true
   #print "Changed agent name ", agent.name


def switchActivitySpace():
  # Switch the activity spaces of agents whose employment status changed
  # Change the times to remain at home, main, rec1 and rec2 and the time needed for travel

  print "Inside switchEmploymentStatus -- Model"

  for citizens as Citizen in self.citizens:
    if citizens.changeEmpStatus == true:
      #print "Index ", String.valueOf(index)
      #print "Agent Name: ", citizens.getName()
      #print "PreChange empStatus ", citizens.employmentStatus
      if citizens.employmentStatus == true:
        #print "My Values should be for unemployed"
        citizens.employmentStatus = false        #Employed becomes unemployed
        citizens.timeHome = citizens.timeUnempHome
        citizens.timeMain = citizens.timeUnempMain
        citizens.rec1 = citizens.timeUnempRec1
        citizens.rec2 = citizens.timeUnempRec2
        citizens.timeTraveling = citizens.timeUnempTraveling
        citizens.changeEmpStatus = false
        #print "Time at Main ", citizens.timeMain
        #print "Time at Main when Unemployed", citizens.timeUnempMain
      else:
        #print "I am employed"
        citizens.employmentStatus = true         #Unemployed becomes employed
        citizens.timeHome = citizens.timeEmpHome
        citizens.timeMain = citizens.timeEmpMain
        citizens.rec1 = citizens.timeEmpRec1
        citizens.rec2 = citizens.timeEmpRec2
        citizens.timeTraveling = citizens.timeEmpTraveling
        citizens.changeEmpStatus = false
        #print "Time at Main ", citizens.timeMain
        #print "Time at Main when Employed", citizens.timeEmpMain

      #print "PostChange ", citizens.employmentStatus
```

```
def initCops():
  print "Inside init cops"
  #Randomly assign the cops to a starting location.

  # Use the Places to get the strnode ##CODE HAS BEEN VERIFIED
  for i in range (self.COPS):
    index = Random.uniform.nextIntFromTo(0, self.NUM_PLACES - 1)
    #print "Index ", index
    cop = Cop()
    cop.setModel(self)

    node = (Place)self.places.get(index)
    #print "FOUND a place " , node.STRCL_

    # Log that cop started at this node
    node.copStart = 1
    cop.setLocation(node)
    cop.setStrnode(node.STRCL_)
    self.cops.add(cop)
```

```
def resetAgentsDaily():
  #print "Inside resetAgentsDaily"

  for citizen as Citizen in self.citizens:
    citizen.atActivity = true
    citizen.atRisk = false
    citizen.moveStatus = false
    citizen.position = 0
    citizen.timeCounter = 0
    citizen.timerHome = 0
    citizen.timerMain = 0
    citizen.timerRec1 = 0
    citizen.timerRec2 = 0
    #print "Counter at reset agent: " + citizen.timerRepeat
```

**def createCitizenTravelOutputFiles():**
```
  print "Inside createCitizenTravelOutputFiles"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Create an output file for model runtime statistics
  # Average number of agents unemployed, average wealth, robbery rate, exposure rate, etc.
  currTick = int(self.getTickCount())
  outFileName = "C:/model_output"+away+"/citizenChar.csv"
  dataWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Tick,NumChangeEmp,NumUnemployed,TotWealth,AvgWealth,RobRate,TotVictims,RepeatVict,Tot
Offen,RepeatOffen,NumExp,PercExposed,NumTravel,PercTraveling,numActiveOffenders,numWaitin
gOffenders,cumDeter,cumIntersect,cumRobberies, cumMultiOffendersAtNode"
  intSize = int(columnNames.length())
  dataWriter.write(columnNames,0,intSize)
  dataWriter.newLine()
  dataWriter.close()
```

**def writeCitizenTravInfotoFiles():**
```
  #print "Inside writeCitizenTravInfotoFiles"
  away = int(self.SOCIETAL_TIMEAWAY*100)

 # Loop through all citizen agents and write out the specified fields
  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    outFileName = "C:/model_output"+away+"/citizens"+agentName+".csv"
    txtWriter = BufferedWriter(FileWriter(outFileName, true))

    tempName = currTick
    home = String.valueOf(citizen.timerHome)
    temp = tempName + "," + home
    main = String.valueOf(citizen.timerMain)
    temp = temp + "," + main
    rec1 = String.valueOf(citizen.timerRec1)
    temp = temp + "," + rec1
    rec2 = String.valueOf(citizen.timerRec2)
    temp = temp + "," + rec2
    travel = String.valueOf(citizen.totTimeTraveling)
    temp = temp + "," + travel
    expose = String.valueOf(citizen.totTimeExposed)
    temp = temp + "," + expose
    vict = String.valueOf(citizen.numVict)
    temp = temp + "," + vict
    offen = String.valueOf(citizen.numOffen)
    temp = temp + "," + offen
    ahome = String.valueOf(citizen.timeHome)
    temp = temp + "," + ahome
    amain = String.valueOf(citizen.timeMain)
    temp = temp + "," + amain
    arec1 = String.valueOf(citizen.timeRec1)
    temp = temp + "," + arec1
    arec2 = String.valueOf(citizen.timeRec2)
    temp = temp + "," + arec2
    atravel = String.valueOf(citizen.timeTraveling)
    temp = temp + "," + atravel

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()
    txtWriter.close()
```

**def writeModelRunData():**

```
  print "Inside writeModelRunData"
  away = int(self.SOCIETAL_TIMEAWAY*100)
  print away

  # CREATE an output file and the buffered writer to write the activity times for each agent to a file
  modelRun = 1
  self.LOG_FILE = "C:/model_output"+away+"/RunDatav"+ modelRun + ".csv"
  txtWriter = BufferedWriter(FileWriter(self.LOG_FILE))

  # Put a model run header
  header =  "Model run: " + modelRun
  intSize = int(header.length())
  txtWriter.write(header,0,intSize)
  txtWriter.newLine()

  # Put a model name in header
  header =  "Model Name: Full_Random Model"
  intSize = int(header.length())
  txtWriter.write(header,0,intSize)
  txtWriter.newLine()

  # Add parameter information
  seed = Random.getSeed()
  nxtLine = "Random Number Seed: " + seed
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Time to reoffend: " + self.REPEAT
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Wealth: " + "Mean " + self.WEALTH_MEAN + ", Standard Deviation " +
self.WEALTH_SD
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  # Close text writer
  txtWriter.close()
```

```
def writeStatistics():
  # Writes out final statistics for all agents in one file to provide summary statistics
  # Aggregate time spent at home, main, rec1, rec2, travel, and exposed.
  # Assigned time to spend at home, main, rec1, rec2, travel.
  # Total number of offenses and victimizations.

  # Create a file
  away = int(self.SOCIETAL_TIMEAWAY*100)
  month = "month"+ self.modelStep/40320

  # Set up to write to file
  outFileName = "C:/model_output"+away+"/statistics_"+month+".csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Agent,timerHome,timerMain,timerRec1,timerRec2,totTimeTraveling,totTimeExposed,numVict,num
Offen,assignHome,assignMain,assignRec1,assignRec2,assignTravel,criminal,wealth"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()

  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    name = String.valueOf(citizen.name)
    home = String.valueOf(citizen.timerHome)
    temp = name + "," + home
    main = String.valueOf(citizen.timerMain)
    temp = temp + "," + main
    rec1 = String.valueOf(citizen.timerRec1)
    temp = temp + "," + rec1
    rec2 = String.valueOf(citizen.timerRec2)
    temp = temp + "," + rec2
    travel = String.valueOf(citizen.totTimeTraveling)
    temp = temp + "," + travel
    expose = String.valueOf(citizen.totTimeExposed)
    temp = temp + "," + expose
    vict = String.valueOf(citizen.numVict)
    temp = temp + "," + vict
    offen = String.valueOf(citizen.numOffen)
    temp = temp + "," + offen
    ahome = String.valueOf(citizen.timeHome)
    temp = temp + "," + ahome
    amain = String.valueOf(citizen.timeMain)
    temp = temp + "," + amain
    arec1 = String.valueOf(citizen.timeRec1)
    temp = temp + "," + arec1
    arec2 = String.valueOf(citizen.timeRec2)
    temp = temp + "," + arec2
    atravel = String.valueOf(citizen.timeTraveling)
    temp = temp + "," + atravel
    acriminal = String.valueOf(citizen.criminalPropensity)
    temp = temp + "," + acriminal
    awealth = String.valueOf(citizen.wealth)
    temp = temp + "," + awealth

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

  # Close the file
  txtWriter.close()
```

**def dataRecorder():**

```
  #print "DATA RECORDER T0 FILE"

  # Writes out model runtime statistics
  # Average number of agents unemployed, average wealth, robbery rate, exposure rate, etc.

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Open the output file and the buffered writer to write the information to a file
  currTick = int(self.getTickCount())
  outFileName = "C:/model_output"+away+"/citizenChar.csv"
  dataWriter = BufferedWriter(FileWriter(outFileName, true))

  # Count number of agents to change employment status
  numChange = 0
  for citizens as Citizen in self.citizens:
    if citizens.changeEmpStatus == true:
      numChange = numChange + 1
      citizens.changeEmpStatus = false

  # Count unemployed agents
  numUnemployed = 0
  numEmployed = 0
  for agent as Citizen in self.citizens:
    if agent.employmentStatus == false:
      numUnemployed = numUnemployed + 1
    elif agent.employmentStatus == true:
      numEmployed = numEmployed + 1
    else:
      print "Employment status not assigned"
  #print "Number unemployed is: ", numUnemployed
  #print "Number employed is: ", numEmployed

  # Calculate average wealth of agents
  totWealth = 0
  for citizens as Citizen in self.citizens:
    totWealth = totWealth + citizens.wealth
  aveWealth = totWealth / self.AGENTS

  # Calculate the robbery rate
  robRate = 0
  robRate = self.totRob / self.AGENTS

  # Count number of agents victimized
  totNumVict = 0
  for citizens as Citizen in self.citizens:
    if citizens.numVict > 0:
      totNumVict = totNumVict + 1
```

```
# Count number of repeat victims
numRepeatVict = 0
for citizens as Citizen in self.citizens:
  if citizens.numVict > 1:
    numRepeatVict = numRepeatVict + 1

# Count number of offenders
totNumOffenders = 0
for citizens as Citizen in self.citizens:
  if citizens.numOffen > 0:
    totNumOffenders = totNumOffenders + 1

# Count number of repeat offenders
numRepeatOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.numOffen > 1:
    numRepeatOffen = numRepeatOffen + 1

# Calculate the number of citizens at risk of victimization
numExp = 0
for citizens as Citizen in self.citizens:
  if citizens.atRisk:
    numExp = numExp + 1
percExp = ((numExp / self.AGENTS) * 100)

# Calculate the number of citizens traveling
numTravel = 0
for citizens as Citizen in self.citizens:
  if citizens.atActivity == false:
    numTravel = numTravel + 1
percTravel = ((numTravel / self.AGENTS) * 100)

# Calculate the number of active offenders (able to offend)
numActiveOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.criminalPropensity and citizens.atRisk and citizens.timerRepeat == 0:
    numActiveOffen = numActiveOffen + 1

# Calculate the number of waiting offenders (not able to offend)
numWaitingOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.criminalPropensity and citizens.atRisk and citizens.timerRepeat > 0:
    numWaitingOffen = numWaitingOffen + 1



# Create a string of each data field to be written to the file
temp = currTick + "," + numChange + "," + numUnemployed + "," + totWealth
temp = temp + "," + aveWealth
temp = temp + "," + robRate
temp = temp + "," + totNumVict
temp = temp + "," + numRepeatVict
temp = temp + "," + totNumOffenders
temp = temp + "," + numRepeatOffen
temp = temp + "," + numExp + ","+ percExp
temp = temp + "," + numTravel + ","+ percTravel
```

```
  temp = temp + "," + numActiveOffen
  temp = temp + "," + numWaitingOffen
  temp = temp + "," + self.totDeter
  temp = temp + "," + self.totIntersect
  #temp = temp + "," + self.totRob + "," + self.multiCriminalsAtNode
  intSize = int(temp.length())
  dataWriter.write(temp,0,intSize)
  dataWriter.newLine()

  #Close the file
  dataWriter.close()
```

**def writeFinalAgents():**

```
  print "Writing Final Agents"
  away = int(self.SOCIETAL_TIMEAWAY*100)
  baseFilePath = "C:/model_output"+away+"/"
  self.writeAgents(self.places, baseFilePath + "strnodes"+away+".shp")
```

**def writeCitizenInfoPaths():**

```
  #print "Inside writeCitizenTravInfotoFiles"
  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Loop through all citizen agents and write out the specified fields
  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    outFileName = "C:/model_output"+away+"/path"+agentName+".csv"
    txtWriter = BufferedWriter(FileWriter(outFileName, true))
    node = citizen.strnode
    temp = node + "," + currTick
    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

    txtWriter.close()
```

**def initCitizensRandom():**

```
  print "Inside initCitizensRandom"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Randomly assign all citizens to a starting location and name them
  for i in range(self.AGENTS):      # Change to 1000 for final model
    p = i + 1
    index = Random.uniform.nextIntFromTo(0, self.NUM_PLACES - 1)
    #print "Index ", index
    citizen = Citizen()
    citizen.setModel(self)
    node = (Place)self.places.get(index)
    #print "Assigned to place ", node.STRCL_
    # Log where citizen started random movement
    node.citiStart = 1
    citizen.setLocation(node)
    citizen.setStrnode(node.STRCL_)
    name = "a" + p
    citizen.setName(name)
    self.citizens.add(citizen)

  # Open log file
  logoutput = self.LOG_FILE
  logWriter = BufferedWriter(FileWriter(logoutput, true))
  logData =  "Log File Name: " + logoutput
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()

  # Randomly assign criminal propensity to 20% of the Citizens
  # Assign 200 agents criminal propensity (criminalPropensity = true)
  for i in range (200):
    index = Random.uniform.nextIntFromTo(0, self.AGENTS - 1)
    agent = (Citizen)self.citizens.get(index)
    #print "Index ", String.valueOf(index)
    #print "Agent Name: "+ agent.getName()+ " is a criminal"
    node = agent.getLocation()

    # Condition to check and make sure citizen was not previously selected to have criminal propensity
    if agent.criminalPropensity == false:
      agent.criminalPropensity = true
      # Log where criminal started random movement
      node.crimStart = 1
    else:
      while agent.criminalPropensity == true:
        index = Random.uniform.nextIntFromTo(0, self.AGENTS - 1)
        agent = (Citizen)self.citizens.get(index)
        node = agent.getLocation()
      # Log where criminal started random movement
      node.crimStart = 1
      agent.criminalPropensity = true
    #print "Agent Name: "+ agent.getName()+ " is a criminal"

    logData = agent.getName() + " ," + " criminal"
    intSize = int(logData.length())
    logWriter.write(logData,0,intSize)
    logWriter.newLine()
    logData = ""

  # Read in files of times to spend at activities for each agent when EMPLOYED
  fileName = "C://Program Files//Repast 3//Agent
Analyst//projects//rob_model//timeSchedule//empTimeAtActivityNodes" + away + ".csv"
  txtreader1 = BufferedReader(FileReader(fileName))

  aName = ""

  # Read each line of the file
  for citizens as Citizen in self.citizens:
    line = txtreader1.readLine()
    tokenizer = StringTokenizer(line, ",")

    # Loop to get all the fields in a particular line
    if (tokenizer.hasMoreTokens()):
      aName = tokenizer.nextToken().trim()
      tEmpHome = tokenizer.nextToken().trim()
      citizens.timeEmpHome = int(tEmpHome)
      tEmpMain = tokenizer.nextToken().trim()
      citizens.timeEmpMain = int(tEmpMain)
      tEmpRec1 = tokenizer.nextToken().trim()
      citizens.timeEmpRec1 = int(tEmpRec1)
      tEmpRec2 = tokenizer.nextToken().trim()
      citizens.timeEmpRec2 = int(tEmpRec2)
      tEmpTravel = tokenizer.nextToken().trim()
      citizens.timeEmpTraveling = int(tEmpTravel)

  # Close the file of activity nodes
  txtreader1.close()

  # Read in files of times to spend at activities for each agent when UNEMPLOYED
  fileName = "C://Program Files//Repast 3//Agent
Analyst//projects//rob_model//timeSchedule//unempTimeAtActivityNodes" + away + ".csv"

  # Read each line of the file
  for citizens as Citizen in self.citizens:
    line = txtreader.readLine()
    tokenizer = StringTokenizer(line, ",")

    # Loop to get all the fields in a particular line
    if (tokenizer.hasMoreTokens()):
      aName = tokenizer.nextToken().trim()
      tUnempHome = tokenizer.nextToken().trim()
      citizens.timeUnempHome = int(tUnempHome)
      tUnempMain = tokenizer.nextToken().trim()
      citizens.timeUnempMain = int(tUnempMain)
      tUnempRec1 = tokenizer.nextToken().trim()
      citizens.timeUnempRec1 = int(tUnempRec1)
      tUnempRec2 = tokenizer.nextToken().trim()
      citizens.timeUnempRec2 = int(tUnempRec2)
      tUnemptravel = tokenizer.nextToken().trim()
      citizens.timeUnempTraveling = int(tUnemptravel)

    #print "Agent:", aName
    #print "Employed Time at Home: ", intEmpHome
    #print "Employed Time at Main: ", intEmpMain
    #print "Employed Time at Rec1: ", intEmpRec1
    #print "Employed Time at Rec2: ", intEmpRec2
    #print "Employed Time Traveling: ", intEmpTravel

    #print "Agent:", aName
    #print "Unemployed Time at Home: ", intUnempHome
    #print "Unemployed Time at Main: ", intUnempMain
    #print "Unemployed Time at Rec1: ", intUnempRec1
    #print "Unemployed Time at Rec2: ", intUnempRec2
```

```
   #print "Unemployed Time Traveling: ", intUnempTravel


  # Close the file of activity nodes
  txtreader.close()

  # SET the employment status for all agents in the model
  # Using a uniform distribution assign 6% of agents to be unemployed
  for i in range (60):
    index = Random.uniform.nextIntFromTo(0, self.AGENTS - 1)
    agent = (Citizen)self.citizens.get(index)
    #print "Index ", String.valueOf(index)

    # Condition to check and make sure citizen was not previously selected to be unemployed
    if agent.employmentStatus == true:
      agent.employmentStatus = false        #Agent is unemployed
    else:
      while agent.employmentStatus == false:
        index = Random.uniform.nextIntFromTo(0, self.AGENTS - 1)
        agent = (Citizen)self.citizens.get(index)
      agent.employmentStatus = false
      #print "Agent Name: "+ agent.getName()+ " is unemployed"

    logData = agent.getName() + " ," + " unemployed"
    intSize = int(logData.length())
    logWriter.write(logData,0,intSize)
    logWriter.newLine()
    logData = ""

  # CREATE an output file of times at each activity and traveling
  # First create the buffered writer to write the activity times for each agent to a file
  outFileName = "C:/model_output" +away+"/timeAtActivityNodes.csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Agent,Home,Main,Rec1,Rec2,Travel,empHome,empMain,empRec1,empRec2,empTravel,unempHo
me,unempMain,unempRec1,unempRec2,unempTravel"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()

  # Assign time schedule to each citizen agent in the model
  for citizens as Citizen in self.citizens:
    if citizens.employmentStatus == true:
      citizens.timeHome = citizens.timeEmpHome
      citizens.timeMain = citizens.timeEmpMain
      citizens.timeRec1 = citizens.timeEmpRec1
      citizens.timeRec2 = citizens.timeEmpRec2
      citizens.timeTraveling = citizens.timeEmpTraveling
      citizens.travelTimeSplit = citizens.timeTraveling / 4    # Allocate equal travel between activities
    elif citizens.employmentStatus == false:
      citizens.timeHome = citizens.timeUnempHome
      citizens.timeMain = citizens.timeUnempMain
      citizens.timeRec1 = citizens.timeUnempRec1
      citizens.timeRec2 = citizens.timeUnempRec2
      citizens.timeTraveling = citizens.timeUnempTraveling
      citizens.travelTimeSplit = citizens.timeTraveling / 4    # Allocate equal travel between activities
    else:
      print "Agent has not been assigned a criminal propensity."

    # Create a string of each data field to be written to the file for later diagnostics
    tempName = String.valueOf(citizens.getName())
    tempHome = String.valueOf(citizens.getTimeHome())
    tempMain = String.valueOf(citizens.getTimeMain())
    tempRec1 = String.valueOf(citizens.getTimeRec1())
    tempRec2 = String.valueOf(citizens.getTimeRec2())
    tempTravel = String.valueOf(citizens.getTimeTraveling())
    tEmpHome = String.valueOf(citizens.getTimeEmpHome())
    tEmpMain = String.valueOf(citizens.getTimeEmpMain())
    tEmpRec1 = String.valueOf(citizens.getTimeEmpRec1())
    tEmpRec2 = String.valueOf(citizens.getTimeEmpRec2())
    tEmpTravel = String.valueOf(citizens.getTimeEmpTraveling())
    tUnempHome = String.valueOf(citizens.getTimeUnempHome())
    tUnempMain = String.valueOf(citizens.getTimeUnempMain())
    tUnempRec1 = String.valueOf(citizens.getTimeUnempRec1())
    tUnempRec2 = String.valueOf(citizens.getTimeUnempRec2())
    tUnempTravel = String.valueOf(citizens.getTimeUnempTraveling())

    # Write values to file
    values =
(tempName+","+tempHome+","+tempMain+","+tempRec1+","+tempRec2+","+tempTravel+","+tEmp
Home+","+tEmpMain+","+tEmpRec1+","+tEmpRec2+","+tEmpTravel+","+tUnempHome+","+tUne
mpMain+","+tUnempRec1+","+tUnempRec2+","+tUnempTravel)
    intSize = int(values.length())
    txtWriter.write(values,0,intSize)
    txtWriter.newLine()

    #print "Agent:", citizens.getName()
    #print "Time Traveling: ", citizens.getTimeTraveling()
    #print "Time at Home: ", citizens.getTimeHome()
    #print "Time at Main: ", citizens.getTimeMain()
    #print "Time at Rec1: ", citizens.getTimeRec1()
    #print "Time at Rec2: ", citizens.getTimeRec2()

  #close the file of activity path nodes
  txtWriter.close()

  # Create new random normal distribution to ASSIGN WEALTH
  Random.createNormal(self.WEALTH_MEAN,self.WEALTH_SD)

  # ASSIGN wealth to agents
  for citizens as Citizen in self.citizens:
    #Get a new random number for each agent
    citizens.wealth = Random.normal.nextInt()
    #print "Name: ", citizens.getName() + " has been assigned Wealth of: "+ citizens.getWealth()

  # Write out initial values for each agent
  # Header line
  logData = "Name, Criminality, Wealth, TimeHome, StartNode"
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = ""
  for citizens as Citizen in self.citizens:
    # Write values
    logData = citizens.getName() + "," + citizens.getCriminalPropensity() + "," + citizens.getWealth() +
"," + citizens.getTimeHome() + "," + citizens.getStrnode()
    intSize = int(logData.length())
    logWriter.write(logData,0,intSize)
    logWriter.newLine()
    logData = ""

  #Close the log file
  logWriter.close()
```

***Place Actions***
(none)

***Citizen Actions***

**def step():**
<u>Java imports</u>
java.lang.Object
java.lang.Double
java.lang.Number
java.lang.Integer
<u>Code</u>

```
 #print "INSIDE CITIZEN STEP"

  # Every citizen agent evaluates their move status, if they are moving they are added to the active
nodes
  # class and are part of the decision to commit a crime.  Then the values (atRisk, atActivity, moving,
and
  # position are set for the next turn.

  # Verification code to check daily start points of agents
  #if self.timeCounter < 2:
    #print "AGENT: ", self.name
    #print "Start Node : ", self.strnode

  # Associate all agents with a current street node so we can test whether they are are atRisk
  # and/or moving.
  self.currentNode = self.strnode

  # Collect all agents who are moving or recreating are at risk and need to be logged
  # at active nodes and put them in the activeNode class.

  #<<CONDITION 1 - Start
  if self.moveStatus == true or self.atRisk == true:

    # ADD an agent to the ActiveNode class.  If there is an ActiveNode agent
    # that exists with a particular strnode value then add the name of the
    # citizen agent to the agentList (an arrayList).  If there is no ActiveNode
    # with the same value as the currentNode then add a new ActiveNode agent and
    # populate the strnode number with the currentNode and add the name of the
    # citizen agent to the agentList (an arrayList).

    #print "Current Node: ", self.currentNode
    #print "The Size is : ", self.model.activeNodes.size()

    # Test to see if this is the first ActiveNode
    nodeisEqual = false
    #<CONDITION 1.1 - Start
    if self.model.activeNodes.size() <> 0:
      for occupied as ActiveNode in self.model.activeNodes:
        if self.currentNode == occupied.strnode:
          occupied.agentList.add(self.name)
          nodeisEqual = true
      #CONDITION 1.1 - End >

    #<CONDITION 1.2 - Start
    if self.model.activeNodes.size() == 0 or nodeisEqual == false:
      newAgent = ActiveNode()
      newAgent.setModel(self.model)
      newAgent.strnode = self.currentNode
      newAgent.agentList = ArrayList()
      newAgent.agentList.add(self.name)
      self.model.activeNodes.add(newAgent)
      #print "First agent of Total Agents: ", self.model.activeNodes.size()
      #print "Inside Assignment: Strnode = ", newAgent.strnode
      #print "ArrayList Value = ", newAgent.agentList.get(0)
    #CONDITION 1.2 - End >


    #print "Current Node as Integer: ", Integer.toString(self.currentNode)
    #print "Home Node: " + self.home +  " Main Node: " + self.main + " Rec 1: " + self.rec1 + " Rec 2:
" + self.rec2
    #CONDITION 1 - End >>

  #print "Home Node: " + self.home +  " Main Node: " + self.main + " Rec 1: " + self.rec1 + " Rec 2: "
+ self.rec2

  # RESET values for next turn.
  # Check to see if currentNode equal to an activity node.
  # If yes, do not move but update time that agent has been at node.  If no, move to next node.
  #<<<CONDITION 2 - Start
  seg1 = self.timeHome + self.travelTimeSplit
  seg2 = self.timeHome + self.travelTimeSplit + self.timeMain
  seg3 = self.timeHome + self.travelTimeSplit + self.timeMain + self.travelTimeSplit
  seg4 = self.timeHome + self.travelTimeSplit + self.timeMain + self.travelTimeSplit + self.timeRec1
  seg5 = self.timeHome + self.travelTimeSplit + self.timeMain + self.travelTimeSplit + self.timeRec1
+ self.travelTimeSplit
  cumulative = seg5 + self.timeRec2

  #print "self.timeHome = " + self.timeHome
  #print "self.timeMain = " + self.timeMain
  #print "self.timeRec1 = " + self.timeRec1
  #print "self.rec1 = " + self.rec1
  #print "self.timeRec2 = " + self.timeRec2
  #print "self.rec2 = " + self.rec2
  #print "seg1 = " + seg1
  #print "seg2 = " + seg2
  #print "seg3 = " + seg3
  #print "seg4 = " + seg4
```

```
#print "seg5 = " + seg5
#print "remainder = " + remainder
#print "self.travelTimeSplit = " + self.travelTimeSplit
#print "self.model.MODEL_DAY = " + self.model.MODEL_DAY

if self.timeCounter <= self.timeHome:
  #print "Time assigned to be at HOME: ", self.timeHome
  #print "Time Spent at Home: ", self.timeCounter
  if self.timeCounter < self.timeHome:
    # Agent is at home
    self.atActivity = true
    self.atRisk = false
    self.moveStatus = false
    #Increment the timer
    self.timeCounter = self.timeCounter + 1
    self.timerHome = self.timerHome + 1
  else:
   # Next step agent leaves home
   self.atActivity = false
   self.atRisk = true
   self.moveStatus = true
   self.timeCounter = self.timeCounter + 1
elif self.timeCounter > seg1 and self.timeCounter <= seg2:
  #print "Time Assigned Main ", self.timeMain
  # "Total Time", self.timeCounter
  if self.timeCounter < seg2:
    # Agent is at Main activity
    self.atActivity = true
    self.atRisk = false
    self.moveStatus = false
    #Increment the timer
    self.timeCounter = self.timeCounter + 1
    self.timerMain = self.timerMain + 1
  else:
    # Next step agent leaves Main activity
    self.atActivity = false
    self.atRisk = true
    self.moveStatus = true
    self.timeCounter = self.timeCounter + 1
    ##print "Agent Leaving Main and moving to position: ", self.position
    self.totTimeTraveling = self.totTimeTraveling + 1
    self.totTimeExposed = self.totTimeExposed + 1
elif self.timeCounter > seg3 and self.timeCounter <= seg4:
  #print "Time Assigned REC1", self.timeRec1
  #print "Total Time: ", String.valueOf(self.timeCounter)
  if self.timeCounter < seg4:
    # Agent is at Rec1
    self.atActivity = true
    self.atRisk = true                #Agents at activities are also at risk
    self.moveStatus = false
    #Increment the timer
    self.timeCounter = self.timeCounter + 1
    self.timerRec1 = self.timerRec1 + 1
    self.totTimeExposed = self.totTimeExposed + 1
  else:
    # Agent is leaving Rec1 next step
```

```
      self.atActivity = false
      self.atRisk = true
      self.moveStatus = true
      self.timeCounter = self.timeCounter + 1
      ##print "Agent Leaving Rec1 and moving to position: ", self.position
      self.totTimeTraveling = self.totTimeTraveling + 1
      self.totTimeExposed = self.totTimeExposed + 1
  elif self.timeCounter > seg5 and self.timeCounter <= cumulative:
    #print "Time assigned Rec2 ", self.timeRec2
    #print "Total time: ", String.valueOf(self.timeCounter)
    if self.timeCounter < cumulative:  # line 135
      # Agent is at Rec2
      self.atActivity = true              #Agents at activities are also at risk
      self.atRisk = true
      self.moveStatus = false
      #Increment the timer
      self.timeCounter = self.timeCounter + 1
      self.timerRec2 = self.timerRec2 + 1
      self.totTimeExposed = self.totTimeExposed + 1
    else:
      # Agent is leaving Rec2 next step
      self.atActivity = false
      self.atRisk = true
      self.moveStatus = true
      self.timeCounter = self.timeCounter + 1
      ##print "Agent Leaving Rec1 and moving to position: ", self.position
      self.totTimeTraveling = self.totTimeTraveling + 1
      self.totTimeExposed = self.totTimeExposed + 1
  else:
    # Agent is traveling
    #print "Agent is Traveling"
    self.atActivity = false
    self.atRisk = true
    self.moveStatus = true
    self.timeCounter = self.timeCounter + 1
    self.totTimeTraveling = self.totTimeTraveling + 1
    #print "Agent time traveling incremented to " + self.totTimeTraveling + " in else of Condition 2.1.1"
    self.totTimeExposed = self.totTimeExposed + 1

    # Select the node the agent will move to in the next turn
    places = self.model.getPlaces()

    #if self.name == "a6" or  self.name == "a7":
    #print "Agent: ", self.name
    #print "Old node: ", self.strnode

    # Identify number of neighbor nodes, generate a random number and use that to pick the next node
    numNeighs = self.location.myNeighbors.size()
    #print "Number of neighbors is: ", numNeighs

    #if self.name == "a6" or  self.name == "a7":
      #print "Number of neighbors is: ", numNeighs

    # Generate a value
    index = Random.uniform.nextIntFromTo(0,numNeighs-1)
    #print "Index picked: ",index
    #if self.name == "a6" or  self.name == "a7":
      #print "Index picked: ",index

    #for node in range (0, numNeighs):
      #print "Neighbor " + node + " is :" + String.valueOf(self.location.getMyNeighbors().get(node))+ "
at position " + index

    # Get the new node and assign it to strnode field
    newNode = self.location.getMyNeighbors().get(index)
    self.strnode = int(String.valueOf(newNode))

    # Add the new node to the agent's nodeList field to track
    #self.nodeList.add(newNode)

    #print self.name + " Move to " + "adjacent strnode: ", self.strnode
    #print self.name + " Move to " + "adjacent strnode: ", self.nodeList.get(0)

    # Do the assignment directly of the strnode to a place
    nodeFL = Float(self.strnode)
    newLocation = (Place)self.model.placeMap.get(nodeFL)
    self.location = newLocation
    #if self.name == "a6" or  self.name == "a7":
      #print "New location: ", self.location.STRCL_

    #CONDITION 2 - END>>>

  #CONDITION 1 - End >
```

**def assignNodeInfo(String tname, int ehome, int emain, int erec1, int erec2, ArrayList ePathNodeList, int uhome, int umain, int urec1, int urec2, ArrayList uPathNodeList):**

```
  # Assigns the variable values read from the files in initCitizens() to the fields in Citizen class
  self.name = tname
  self.home = ehome
  self.empHome = ehome
  self.empMain = emain
  self.empRec1 = erec1
  self.empRec2 = erec2
  self.empPathNodes = ePathNodeList
  self.unempHome = uhome
  self.unempMain = umain
  self.unempRec1 = urec1
  self.unempRec2 = urec2
  self.unempPathNodes = uPathNodeList
  self.currentNode = self.home
```

**def payCitizens():**

```
  #print "Inside Pay Citizens"

  # Each employed citizen gets paid at designated intervals

  if self.employmentStatus == true:
    #print "Agent Name: ",self.name
    #print "Agent Old Wealth: ", self.wealth
    self.wealth = self.wealth + 5

    #print "Agent New Wealth: ", self.wealth
```

***Active Node Actions***
(none)

***Cop Actions***

**def step():**
<u>Java imports</u>
java.lang.Object
java.lang.String
uchicago.src.sim.util.SimUtilities
java.util.Arrays
java.util.List
<u>Code</u>

```
  # Every cop moves with each model tick
  places = self.model.getPlaces()
  #print "Old node: ", self.strnode

  # Shuffle the adjacent nodes of the Place where the cop is located
  # Identify number of neighbor nodes
  numNeighs = self.location.myNeighbors.size()
  maxValue = numNeighs-1

  # Generate a value
  index = Random.uniform.nextIntFromTo(0,numNeighs -1)

  #print "Move to index: " + index

  # Verification code
  #for node in range (0, numNeighs):
    #print "Neighbor " + node + " is :" + String.valueOf(self.location.getMyNeighbors().get(node))

  # Get the new node and assign it to strnode field
  # (can't just use index because index and strnode do not correspond)
  newNode = self.location.getMyNeighbors().get(index)
  self.strnode = int(String.valueOf(newNode))

  #print "New strnode: ", self.strnode

  # Do the assignment directly of the strnode to a place
  nodeFL = Float(self.strnode)
  newLocation = (Place)self.model.placeMap.get(nodeFL)
  self.location = newLocation
  #print "New location: ", self.location.STRCL_
```

***Sequence Graph***

totalRobberies
return self.totRob

totalDeterred

```
return self.totDeter

totalIntersect
return self.totIntersect
```

*Appendix 5: Street Robbery Model: Activity Space Version Code*

***Street Robbery Activity Space Actions***

**def initAgents():**

Java imports

```
uchicago.src.simbuilder.util.MessageDisplay
java.lang.String
anl.repast.gis.data.dbf.DBFReader
anl.repast.gis.data.dbf.JDBField
java.Array
java.util.Vector
java.util.List
java.lang.Object
java.util.ArrayList
uchicago.src.sim.util.Random
java.io.PrintWriter
```

Code

```
 print "Inside initAgents"
 if (self.messageDisplay == None):
   self.messageDisplay = MessageDisplay()
   self.messageDisplay.display("Street Robbery Messages")
 else:
   self.messageDisplay.clear()

 # Explicitly set the random number generator seed and initialize Random distributions
 # Create RNG and set seed
 mtRNG = MersenneTwister(100)
 Random.setSeed(100)
 Random.createUniform()

 # Create log file for model run
 self.writeModelRunData()

 # Initialize model level variables
 self.initModel()

 # Process the street nodes for use in the model
 self.setupPlaces()

 # Initialize the activity spaces of agents
 self.initActivitySpaces()

 # Initialize the activity spaces of agents
 self.initCitizens()

 # Create a normal random number distribution for agent movement
 self.NORM_TRAVEL = Normal(6,1,mtRNG)

 # Create output files for analysis
 self.createCitizenTravelOutputFiles()

 # Check to make sure values in shapefile fields are zero
```

```
  for node as Place in self.places:
    if node.totalRob > 0 or node.totalVisit > 0 or node.totPrevent > 0:
      print "WARNING:  Shapefile had non-zero values in counter fields"

  # Initialize the cop agents
  self.initCops()
```

**def updateDisplay():**

```
  #print "Inside updateDisplay"
  self.updateGISDisplay()
```

**def writeAgents():**

```
  #print "Inside writeAgents-Model level"
  baseFilePath = ".\\projects\\rob_model\\shapefiles\\"
  self.writeAgents(self.places, baseFilePath + "strnodes2.shp")
```

**def setupPlaces():**

Java imports

```
java.io.BufferedReader
java.io.FileReader
java.util.StringTokenizer
```

Code

```
  print "Inside setupPlaces"

  # Put Places in a HashMap where the key is the strnode-id
  # Creates the map
  self.placeMap = LinkedHashMap()

  # Add the places to the hashmap
  for currentPlace as Place in self.places:
    specNode = "0"
    specNode = String.valueOf(currentPlace.getSTRCL_())
    specNodeNew = Float(specNode)
    self.placeMap.put(specNodeNew, currentPlace)
    #print "PLACE node info: ", specNodeNew
    currentPlace.setMyNeighbors(ArrayList())

  # Read the neighbors file and set each nodes neighbors.
  # The neighbors files lists the active node and the neighboring
  # nodes of that active node.  The map created above is used to
  # get the neighbors for each active node.

  fileName = "./projects/rob_model/neighborFiles/nodenghbrs.csv"
  reader = BufferedReader(FileReader(fileName))
  line = reader.readLine()

  while(line):
    tokenizer = StringTokenizer(line, ",")
    if(tokenizer.hasMoreTokens()):
      activeNode = tokenizer.nextToken().trim()
      actNodeObject = Float(activeNode)
      currentPlace = (Place)self.placeMap.get(actNodeObject)
    #print "Current variable ", activeNode #prints out the variable strcl_
    #print "Current node from place object:  ", currentPlace.getSTRCL_()
    nghs = currentPlace.getMyNeighbors()
    while (tokenizer.hasMoreTokens()):
      ngh = tokenizer.nextToken()
      currentPlace.myNeighbors.add(ngh)
      #print "Neighbor node ", ngh
  # Read the line
  line = reader.readLine()
 # Close the reader
 reader.close()

 #This code enables verification that the myNeighbors array has the correct values
 for currentPlace as Place in self.places:
   #print "Streetnode: ", node.strcl_
   if currentPlace.getMyNeighbors() == None:
     print "Neighbor arraylist is empty for node " + currentPlace.strcl_
```

**def showMessage(String message):**

<u>Java imports</u>

javax.swing.JOptionPane

<u>Code</u>

```
 print "Inside showMessage"
 JOptionPane.showMessageDialog((JComponent)None, message)
```

**def incrementModel():**

```
 #print "Inside incrementModel"

  # Increment the modelStep field
 #if self.modelStep < self.MODEL_DAY:     #1,440
 #if self.modelStep < self.MODEL_WEEK:
 #if self.modelStep <  self.MODEL_YEAR:       #525,600
 if self.modelStep < 40320:      # month is 40,320
   self.modelStep = self.modelStep + 1
 else:
   self.writeFinalAgents()
   for node as Place in self.places:
     node.totalVisit = 0
     node.totalRob = 0
     node.totPrevent = 0
     node.copStart = 0
     node.citiStart = 0
     node.crimStart = 0
   self.writeAgents()
   self.writeStatistics()
   self.dataRecorder()
   print "YEAR OVER"
   self.stop()
 #print "MODEL STEP = ", self.modelStep

 # ActiveNode - call a method to write out a file of the nodes and their
 # associated agents at each step
 #print "Called writeOccupiedNodes"
 #self.writeOccupiedNodes()

 # Write out citizen position
 #self.writeCitizenInfoPaths()

 # Make the decision to commit a crime
 # print "Total Active Nodes: ", self.activeNodes.size()
 self.decideRob()

 # Clear the agents from the activeNodes class
 self.activeNodes.clear()
 #print "Total active nodes after clear: ", self.activeNodes.size()

  # Increment the timers for agents with criminal propensity
 for citizen as Citizen in self.citizens:
   if citizen.criminalPropensity == true:
     if citizen.timerRepeat > 0 and citizen.timerRepeat < self.REPEAT:
       citizen.timerRepeat = citizen.timerRepeat + 1
       #print "Agent " + citizen.name + " repeat timer incremented to: " + citizen.timerRepeat
     elif citizen.timerRepeat == self.REPEAT:
       #print "REPEAT value: " + self.REPEAT
       citizen.timerRepeat = 0
       #print "Agent " + citizen.name + " timer reset to 0: " + citizen.timerRepeat
     else:
       citizen.timerRepeat = citizen.timerRepeat

 #print "TOTAL Robberies in society: ", self.totRob
```

**def initModel():**

Java imports

```
cern.jet.random.*
cern.jet.random.engine.MersenneTwister
uchicago.src.sim.util.Random
cern.jet.random.Normal
```

Code

```
 print "Inside initModel"

 # Open log file
 logoutput = self.LOG_FILE
 logWriter = BufferedWriter(FileWriter(logoutput, true))

 # Set static field values for model run
 self.modelStep = 0
 self.MODEL_HOUR = 60  #360 steps per hour, Travel occurs at 6 steps per minute
 self.MODEL_DAY = (24 * self.MODEL_HOUR)
 self.MODEL_WEEK = (7 * self.MODEL_DAY)
 self.MODEL_YEAR = (365 * self.MODEL_DAY)

# Print to file to document model run
 temp =  self.SOCIETAL_TIMEAWAY *100
 logData = "Experimental condition: " + temp + "% time spent away from home" + "::::" + "Number
of Agents in model " + self.AGENTS
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
 logData = ""
 logData = "Number of cops: " + self.COPS
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
 logData = ""
 logData = "Limit on Repeat Offending: " + self.REPEAT
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
 logData = ""

 #Close the log file
 logWriter.close()


def initActivitySpaces():
 print "Inside initActivitySpaces"
 print "Number of Citizen Agents in model: " + self.citizens.size()

 ##Read two types of files to get the activity nodes and the path nodes for each
 # Civilian agent and populate the Civilian agents
 # Each civilian agent has a list of activity nodes in a separate file:
 # Order of activity nodes in file:
  # Node 0:  HOME
  # Node 1:  MAIN
  # Node 2:  REC1
  # Node 3:  REC2
 # FILE 1:  Read in the list of activity nodes for each agent from individual files and
 ##assign to the generic agent class of ActivityNode (group name: activityNodes).

 # Initialize variables
 theAgentName = "0"
 theEmpHome = 0
 theEmpMain = 0
 theEmpRec1 = 0
 theEmpRec2 = 0
 theUnempHome = 0
 theUnempMain = 0
 theUnempRec1 = 0
 theUnempRec2 = 0
 strNode = 0
 j = 0

 #outside loop to run through files for activity space information files (counter is used for both files)
 for i in range (1000):
  j = i + 1
  theAgentName = "a" + String.valueOf(j)
  fileName = "./projects/rob_model/activityNodeFiles/" + theAgentName + "empnodes" + ".csv"
  txtreader = BufferedReader(FileReader(fileName))
  line = txtreader.readLine()
  lineNumber = 0

  # Loop to read one line at a time, each file in turn
  while (line):
    tokenizer = StringTokenizer(line, ",")
    #counter for the four records in each actNode file
    if lineNumber > 3:
      lineNumber = 0

    # Loop to get all the fields in a particular line
    if (tokenizer.hasMoreTokens()):
      strnode = tokenizer.nextToken().trim()
      stringNode = int(strnode)
      x = tokenizer.nextToken().trim()
      y = tokenizer.nextToken().trim()
      #Conditions to assign strnode number in first line of file
      #to Home, second record to Main ....etc.
      #print "Strnode :", strnode
      if lineNumber == 0:
        theEmpHome = stringNode
      elif lineNumber == 1:
        theEmpMain = stringNode
      elif lineNumber == 2:
        theEmpRec1 = stringNode
      else:
        theEmpRec2 = stringNode

      lineNumber = lineNumber + 1
      line = txtreader.readLine()

  # Close the file of activity nodes
  txtreader.close()

  # FILE 2: Each EMPLOYED civilian agent has a list of path nodes they travel to get to
  # their activity nodes (one file for each agent):

  # Read in the list of path nodes for each agent from individual files and
  # assign to the generic agent class.
  shpFilePath = "./projects/rob_model/pathFiles/" + theAgentName + "empnodespath" + ".dbf"

  # Create reader and get first field
  reader = DBFReader(shpFilePath)
  fieldCount = reader.getFieldCount()

  # Declaring the list here creates a new one for each file read.
  theEmpPathNodeList = ArrayList()
  while (reader.hasNextRecord()):
    theObject = reader.nextRecord()
    objList = Arrays.asList(theObject)
    #print "Field value ", objList.get(0).toString()
    theEmpPathNodeList.add(objList.get(0))


# UMEMPLOYED -- Read the file of unemployed activity nodes for each agent
  #theAgentName = "a" + String.valueOf(j)
  fileName = "./projects/rob_model/activityNodeFiles/" + theAgentName + "unempnodes" + ".csv"
  txtreader = BufferedReader(FileReader(fileName))
  line = txtreader.readLine()
  lineNumber = 0
```

```
# Loop to read one line at a time, each file in turn
while (line):
  tokenizer = StringTokenizer(line, ",")
  #counter for the four records in each actNode file
  if lineNumber > 3:
    lineNumber = 0

  #Loop to get all the fields in a particular line
  if (tokenizer.hasMoreTokens()):
    strnode = tokenizer.nextToken().trim()
    stringNode = int(strnode)
    x = tokenizer.nextToken().trim()
    y = tokenizer.nextToken().trim()
    #Conditions to assign strnode number in first line of file
    #to Home, second record to Main ....etc.
    #print "Strnode :", strnode
    if lineNumber == 0:
      theUnempHome = stringNode
    elif lineNumber == 1:
      theUnempMain = stringNode
    elif lineNumber == 2:
      theUnempRec1 = stringNode
    else:
      theUnempRec2 = stringNode

    lineNumber = lineNumber + 1
    line = txtreader.readLine()

# Close the file of activity nodes
txtreader.close()

# FILE 2: Each civilian agent has a list of UNMEMPLOYED path nodes they travel to get to
# their activity nodes (one file for each agent):

# Read in the list of path nodes for each agent from individual files and
# assign to the generic agent class.
shpFilePath = "./projects/rob_model/pathFiles/" + theAgentName + "unempnodespath" + ".dbf"

# Create reader and get first field
reader = DBFReader(shpFilePath)
fieldCount = reader.getFieldCount()

# Declaring the list here creates a new one for each file read.
theUnempPathNodeList = ArrayList()
while (reader.hasNextRecord()):                    # while reads until there are no more records
  theObject = reader.nextRecord()
  objList = Arrays.asList(theObject)
  #print "Field value ", objList.get(0).toString()
  theUnempPathNodeList.add(objList.get(0))

# Create an individual citizen agent called citizenSet within the loop to read the files in the directory
citizenSet = Citizen()
citizenSet.setModel(self)

# Assign the values from the file to the agent variables  #uncomment after testing
```

```
    citizenSet.assignNodeInfo(theAgentName, theEmpHome, theEmpMain, theEmpRec1, theEmpRec2,
theEmpPathNodeList, theUnempHome, theUnempMain, theUnempRec1, theUnempRec2,
theUnempPathNodeList)

    # Add an agent to the ActivityNode class with the field values from above
    self.citizens.add(citizenSet)                  #uncomment after testing
```

**def initCitizens():**

```
  print "Inside initCitizens"

  # Open log file
  logoutput = self.LOG_FILE
  logWriter = BufferedWriter(FileWriter(logoutput, true))
  logData =  "Log File Name: " + logoutput
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = "Agents with Criminal Propensity: "
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
  logWriter.newLine()
  logData = ""

  # Create new random normal distribution to ASSIGN WEALTH
  Random.createNormal(self.WEALTH_MEAN,self.WEALTH_SD)

  # ASSIGN wealth to agents
  for citizens as Citizen in self.citizens:
    #Get a new random number for each agent
    citizens.wealth = Random.normal.nextInt()
    #print "Name: ", citizens.getName() + " has been assigned Wealth of: "+ citizens.getWealth()

  # Randomly assign criminal propensity to 20% of the Citizens
  # Assign 200 agents criminal propensity (criminalPropensity = true)
  for i in range (200):
    index = Random.uniform.nextIntFromTo(0, 999)
    agent = (Citizen)self.citizens.get(index)
    #print "Index ", String.valueOf(index)
    #print "Agent Name: "+ agent.getName()+ " is a criminal"

    # Condition to check and make sure citizen was not previously selected to have criminal propensity
    if agent.criminalPropensity == false:
      agent.criminalPropensity = true
    else:
      while agent.criminalPropensity == true:
        index = Random.uniform.nextIntFromTo(0, 999)
        agent = (Citizen)self.citizens.get(index)
      agent.criminalPropensity = true
     #print "Agent Name: "+ agent.getName()+ " is a criminal"

    logData = agent.getName() + " is a criminal."
    intSize = int(logData.length())
    logWriter.write(logData,0,intSize)
    logWriter.newLine()
    logData = ""
```

```
# SET the employment status for all agents in the model
# Using a uniform distribution assign 6% of agents to be unemployed
for i in range (60):
 index = Random.uniform.nextIntFromTo(0, 999)
 agent = (Citizen)self.citizens.get(index)
 #print "Index ", String.valueOf(index)

 # Condition to check and make sure citizen was not previously selected to be unemployed
 if agent.employmentStatus == true:
   agent.employmentStatus = false        #Agent is unemployed
 else:
   while agent.employmentStatus == false:
     index = Random.uniform.nextIntFromTo(0, 999)
     agent = (Citizen)self.citizens.get(index)
   agent.employmentStatus = false
 #print "Agent Name: "+ agent.getName()+ " is unemployed"

 logData = agent.getName() + " is a criminal."
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
 logData = ""


# SET the activity node and activity path to be used in assigning time spent at activities
for citizens as Citizen in self.citizens:
  nodeFL = Float(citizens.empHome)   # empHome and UnempHome are the same value
  node = (Place)self.placeMap.get(nodeFL)
  #print "Street node home: " + citizens.empHome
  node.citiStart = 1
  #print "Citizen starts at node: " + node.STRCL_
  if citizens.employmentStatus == true:
    citizens.main = citizens.empMain
    citizens.rec1 = citizens.empRec1
    citizens.rec2 = citizens.empRec2
    citizens.pathNodes = citizens.empPathNodes
    citizens.setLocation(node)
    citizens.setStrnode(node.STRCL_)
  else:
    citizens.main = citizens.unempMain
    citizens.rec1 = citizens.unempRec1
    citizens.rec2 = citizens.unempRec2
    citizens.pathNodes = citizens.unempPathNodes
    citizens.setLocation(node)
    citizens.setStrnode(node.STRCL_)
  if citizens.criminalPropensity == true:
    node.crimStart = 1

  #print "Name ", citizens.name
  #print "Status ", citizens.employmentStatus
  #print "Main ", citizens.main
  #print "UnempMain ", citizens.unempMain
  #print "EmpMain ", citizens.empMain

###Randomly assign time to stay at home to each agent
```

```
  ## Five experimental conditions:
  #  30% timeAway = 70% of time at home (1008 minutes)
  #  40% timeAway = 60% of time at home
  #  50% timeAway = 50% of time at home
  #  60% timeAway = 40% of time at home
  #  70% timeAway = 30% of time at home
  # Create a new random number generator to create a normal distribution
  # with a mean of 70 and SD of 10.  Since there is no literature on the
  ##split before and after midnight, randomly assign split.


  # CREATE an output file of times at each node and traveling
  # First create the buffered writer to write the activity times for each agent to a file

  #Commented out code split times at home between start and end node
  away = int(self.SOCIETAL_TIMEAWAY*100)
  outFileName = "C:/model_output" +away+"/timeAtActivityNodes.csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames = "Agent,Home,Main,Rec1,Rec2,Travel"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()

  # Normal distribution to assign amount of time to spend at home
  # Mean is the society mean and SD is ten percent of the mean
  societyPercHome = 1 - self.SOCIETAL_TIMEAWAY
  standardDeviation = (self.MODEL_DAY * societyPercHome)*.10
  meanTimeHome = self.MODEL_DAY * societyPercHome

  # Create a normal distribution with specified mean and sd
  Random.createNormal(meanTimeHome, standardDeviation)

  # Allocate the times to remain at home, main, rec1 and rec2 and the time needed for travel
  timeLeftForRec = 0
  for citizens as Citizen in self.citizens:

    # Get a new random number for each agent
    timeAtHome = Random.normal.nextInt()
    #print "Random time at home: ", totTimeAtHome

    # Get a new proportion to split time not at home
    randSplit = Random.uniform.nextDoubleFromTo(.1, .9)

    # Loop through the agents and find out how many nodes are in the activityPath
    totSteps = citizens.getPathNodes().size()
    #print "Total Steps: ", totSteps

    # Convert the travel time in steps to travel time in minutes
    citizens.timeTraveling = totSteps / 6
    timeAfterTravel = self.MODEL_DAY - citizens.timeTraveling

    # While loop to check for negative numbers and reset timeAtHome until it is less than
timeAfterTravel.
    # This ensures the agents have time to spend at each activity node besides Home.
    # while timeAfterTravel < totTimeAtHome:
    while timeAfterTravel < timeAtHome:
```

```
    timeAtHome = Random.normal.nextInt()

  extra = timeAtHome - timeAfterTravel
  timeAfterTravel_Home = timeAfterTravel - timeAtHome
  citizens.timeHome = timeAtHome

  # Checks employment status before distributing time to main and recreations places
  if citizens.employmentStatus:    #Employed
    citizens.timeMain = (timeAfterTravel_Home/2)
    citizens.timeRec1 = int(citizens.timeMain * randSplit)
    citizens.timeRec2 = citizens.timeMain - citizens.timeRec1
  else:                          #Unemployed
    citizens.timeMain = int(timeAfterTravel_Home * randSplit)
    timeLeftForRec = timeAfterTravel_Home - citizens.timeMain
    citizens.timeRec1 = int(timeLeftForRec * randSplit)
    citizens.timeRec2 = timeLeftForRec - citizens.timeRec1
    #citizens.timeMain = (timeAfterTravel_Home/3)
    #citizens.timeRec1 = int(citizens.timeMain * randSplit)
    #citizens.timeRec2 = citizens.timeMain - citizens.timeRec1


  #create a string of each data field to be written to the file
  tempName = String.valueOf(citizens.getName())
  tempHome = String.valueOf(citizens.getTimeHome())
  tempMain = String.valueOf(citizens.getTimeMain())
  tempRec1 = String.valueOf(citizens.getTimeRec1())
  tempRec2 = String.valueOf(citizens.getTimeRec2())
  tempTravel = String.valueOf(citizens.getTimeTraveling())
  intSize = int(values.length())
  txtWriter.write(values,0,intSize)
  txtWriter.newLine()

  #print "Agent:", citizens.getName()
  #print "Time Traveling: ", citizens.getTimeTraveling()
  #print "Time at Home: ", citizens.getTimeHome()
  #print "Time at Main: ", citizens.getTimeMain()
  #print "Time at Rec1: ", citizens.getTimeRec1()
  #print "Time at Rec2: ", citizens.getTimeRec2()

 #close the file of activity path nodes
 txtWriter.close()


 # Write out initial values for each agent
 # Header line
 logData = "Name, Criminality, Wealth_Start, TimeHome, StartNode"
 intSize = int(logData.length())
 logWriter.write(logData,0,intSize)
 logWriter.newLine()
 logData = ""
 for citizens as Citizen in self.citizens:
  # Write values
  logData = citizens.getName() + "," + citizens.getCriminalPropensity() + "," + citizens.getWealth() +
"," + citizens.getTimeHome() + "," + citizens.getStrnode()
  intSize = int(logData.length())
  logWriter.write(logData,0,intSize)
```

```
    logWriter.newLine()
    logData = ""

  #Close the log file
  logWriter.close()

  #for citizens as Citizen in self.citizens:
   #print "Name: " + citizens.getName()
   #print "Home: " + citizens.getHome()
   #print "Main: " +citizens.getMain()
   #print "Rec1: " +citizens.getRec1()
   #print "Rec2: " +citizens.getRec2()
   #print "Employed?: " +citizens.getEmploymentStatus()
```

**def decideRob():**

Java imports

```
java.lang.Object
java.lang.String
uchicago.src.sim.util.SimUtilities
java.util.Arrays
java.util.List
```

Code

```
  #print "Inside decideRob"

  # Open log file
  logoutput = self.LOG_FILE
  logWriter = BufferedWriter(FileWriter(logoutput, true))
  #data = "Inside Decide Rob Action"
  #intSize = int(data.length())
  #logWriter.write(data,0,intSize)
  #logWriter.newLine()
  #data = ""

  # Check each ActiveNode for list of agents
  # Logical error check
  if self.activeNodes.size() > self.AGENTS:
    data = "Too Many Active Nodes during step: " + self.modelStep
    intSize = int(data.length())
    logWriter.write(data,0,intSize)
    logWriter.newLine()
    data = ""
  #else:
    #data = "Number of Active Nodes in: " + self.modelStep + " is: "  + self.activeNodes.size()
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

  # Loop through the nodes with citizens and make the decision to commit a robbery
  for occupied as ActiveNode in self.activeNodes:
    # Check agents at each of the active nodes
    #print "The Node being evaluated is: ", occupied.strnode

    #################
    #data = "Street Node " + occupied.strnode
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    # Initialize variables in action
    numAgentsAtNode = occupied.getAgentList().size()
    #data = "Number of Agents at Node (from size of agent list field): " + numAgentsAtNode
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    numCrimAtNode = 0
    numAgentAtRisk = 0
    numCriminals = 0
    offenderAtNode = false
    curStreetNode = (Place)self.places.get(0)
    #print "The default curStreetNode: ", curStreetNode.STRCL_
    curAgent = (Citizen)self.citizens.get(0)
    targetAgent = (Citizen)self.citizens.get(0)    #used to initialize target variable
    criminalAgent = (Citizen)self.citizens.get(0)
    copPresent = false
    robbery = true
    crimWealth = 0
    evalWealth = 0
    targetWealth = 0
    suitability = 0
    targetSet = false

    #################
    #data = "START VALUES: Criminal Agent-- " + criminalAgent.name + "Current Agent-- " +
curAgent.name + "Target agent- " + targetAgent.name
    #intSize = int(data.length())
    #logWriter.write(data,0,intSize)
    #logWriter.newLine()
    #data = ""

    # Log presence of agents on street node
    # Retrieving the place by converting to a float object
    occupiedObject = Float(occupied.strnode)
    currentPlace = (Place)self.placeMap.get(occupiedObject)
    #print "NEW Place node:", currentPlace.getSTRCL_()
    #print "Number of agents at Node: ", numAgentsAtNode
    #self.messageDisplay.addAlert("There are "+ numAgentsAtNode + " citizens at " +
occupied.strnode)

    # Log fact that agents visited a node in the shapefile
    if (currentPlace != None):
      currentPlace.totalVisit = currentPlace.totalVisit + numAgentsAtNode
      #currentPlace.visits = currentPlace.visits + numAgentsAtNode
      #print "NEW Number of Visits: ", currentPlace.totalVisit
    else:
      print "Unable to log visit at strnode: " + occupied.strnode + " during tick: " + self.modelStep
    #Loop through all the cops to find out if there is a cop at node
    for copAtNode as Cop in self.cops:
      copPlace = copAtNode.getLocation()
      #When you find a cop at the place break out of loop and calculate variable
      if copPlace == currentPlace:
        #print "Cop at node: ", copAtNode.location.STRCL_
        copPresent = true
        break
      else:
        copPresent = false

    ##############################DEBUG
    #if copPresent == true:
      #data = "Cop is at node! "
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
    #if copPresent == true:
      #print "Cop! at node "+ occupied.strnode

    #Only evaluate nodes that have more than one citizen and there is no cop present
    if numAgentsAtNode > 1:              #Change to > 1 for final testing

      #################  DEBUG
      #data = "Street Node " + occupied.strnode
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
      ##############################
      #data = "Number of Agents at Node is: " + numAgentsAtNode
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""

      # Calculate the level of GUARDIANSHIP
      guardianship = (numAgentsAtNode - 2) +
Random.uniform.nextIntFromTo(self.MIN_GUARDIANSHIP,self.MAX_GUARDIANSHIP)
      #print "Guardianship is: ", guardianship

      # Outside loop that checks each of the agents at a particular node using the citizen name
      # Use code to randomly shuffle the agents at a node so they have an equal chance of being
      # selected first and thus are not always evaluated in the same order.

      #for position in range(0, numAgentsAtNode):
        #print "Original Order: Node--" + occupied.strnode + " Position " + position + ", "+
String.valueOf(occupied.getAgentList().get(position))

      # Create a distribution using number of agents at node
      maxValue = numAgentsAtNode-1

      # Create arraylist variables
      # Array to hold randomly shuffled agents
      randList = ArrayList()
      # List of array positions that have been used
      strList = ArrayList()
      foundIt = false
      #print numAgentsAtNode
      # Outside while to create a new list of all the agents at the node in a new order

      while randList.size() < numAgentsAtNode:
        # Generate a random number
        foundIt = false
        #index = shuffleDist.nextInt()
        index = Random.uniform.nextIntFromTo(0,numAgentsAtNode-1)
        indexStr = String.valueOf(index)
        #print "Original index generated is: " + index

        ##############################
        #data = "Index value: " + index
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logwriter.newLine()
        #data = ""

        # If this is the first agent generated then add it to the new randList array, otherwise check to see if
        # the index has already been used
        if strList.size() >= 1:
          for p in range (0, strList.size()):
            if String.valueOf(strList.get(p)) == indexStr:
              foundIt = true
              break

        if foundIt == false:
          agent = occupied.AgentList.get(index)
          randList.add(agent)
          strList.add(indexStr)
          #print "randList size is ", randList.size()
          #print "New size of list of index numbers is " + strList.size()

      # Code to verify new order
      #for position in range(0, numAgentsAtNode):
        #print "New order at Node: " + occupied.strnode + " position " + position + ", " +
String.valueOf(randList.get(position))

      for i in range (0,numAgentsAtNode):
        #Bunch of code that get the agent name (e.g. a1) and then strips off the first character
        #and pulls the correct Citizen agent using the agent name
        fullName = randList.get(i)
        fullStrName = String.valueOf(fullName)
        partName = fullStrName.substring(1)
        #print "Agent: " + partname + " in agentList"
        # Use the agent's name to find the index number of correct Citizen agent
        index = int(partName) - 1
        curAgent = (Citizen)self.citizens.get(index)
        #print "Citizen in agentList: ", curAgent.name
        #print "Current agent: " + curAgent.name + " is criminal? " + curAgent.criminalPropensity

        ##################################
```

```
        #data = "Loop through current agents to find Criminal: " + i + "," +
String.valueOf(randList.get(i)) + "," + "criminal: " + curAgent.criminalPropensity + "," + " at risk?: " +
curAgent.atRisk
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""

        # Identifies if any of the agents have criminal propensity and are 'atRisk' and selects the
        # first one it finds to be the active criminal in this interaction
        if curAgent.criminalPropensity == true and curAgent.atRisk == true:
          criminalAgent = curAgent
          if criminalAgent.timerRepeat == 0:
            offenderAtNode = true
          #else:
            #print "Agent " + criminalAgent.name + " Offender unable to offend yet"
          break        #go directly to next if statement (if offenderAtNode == true:)

      ##################################DEBUG
      #data = "Criminal in interaction: " + criminalAgent.name + "," + "timer: " +
criminalAgent.timerRepeat
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""


      # Loop that uses formulas to evaluate guardianship and target suitability
      if offenderAtNode == true:
        #print "Offender: " + criminalAgent.name + " is at Node and evaluating opportunity"

        # Find out how many civilians are 'at risk' and
        # which 'at risk' civilian at the active node has the most wealth
        for i in range (0,numAgentsAtNode):
          # Get the first agent in the randomly ordered list
          fullName = randList.get(i)
          fullStrName = String.valueOf(fullName)
          partName = fullStrName.substring(1)
          # Use the agent's name to find the index number of correct Citizen agent
          index = int(partName) - 1
          evalAgent = (Citizen)self.citizens.get(index)
          evalWealth = evalAgent.wealth
          crimWealth = criminalAgent.Wealth
          #print "Criminal's Wealth: ", crimWealth
          #print "Evaluated agent: ", evalAgent.name
          #print "Evaluated agent's wealth: ", evalWealth

          # Counter for number of criminals at node
          if evalAgent.criminalPropensity == true:
            numCrimAtNode = numCrimAtNode + 1


          ##############################DEBUG
          #data = "Evaluate Wealth for eval agent: " + evalAgent.name + " has wealth of: " + evalWealth
+ " CrimAgent: " + criminalAgent.name + "has wealth of: " + crimWealth
```

```
        #data = "Eval agent: " + "," + evalAgent.name + "," + "Wealth: " + "," + evalWealth + "
CrimAgent: " + criminalAgent.name + "has wealth of: " + crimWealth
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""

        # Counter for number of agents at node who are 'at risk' of being robbed (only is counted
        # if there is an offender at the node)
        if (evalAgent.atRisk == true) and (evalAgent.name != criminalAgent.name):
          numAgentAtRisk = numAgentAtRisk + 1

        ##############################
        #data = "Number of agents at risk: " + numAgentAtRisk
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""

        # Identify the 'at risk' agent with the most wealth
        if criminalAgent.name != evalAgent.name:
          ##############################
          #data = "Comparing " + criminalAgent.name + " with " + crimWealth + " to evaluated agent: "
+ evalAgent.name + " with " + evalWealth
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

          #print "Comparing " + criminalAgent.name + " with " + crimWealth + " to evaluated agent: " +
evalAgent.name + " with " + evalWealth
          if (crimWealth <= evalWealth) and (evalAgent.atRisk == true):
            if evalWealth > targetWealth:
              targetWealth = evalWealth
              #targetAgent = (Citizen)self.citizens.get(index)
              targetAgent = evalAgent
              targetSet = true
              #print "Current Agent with highest wealth ," + targetAgent.name

      ######################## DEBUG
      #data = "Identity of selected targetAgent: " + "," + targetAgent.name + "," +
targetAgent.criminalPropensity
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()
      #data = ""
      #print "Number of agents at risk", numAgentAtRisk

      ######################Print summary
      #if numCrimAtNode >= 2:
        #data = "Two or more criminals present at node during Model Step: " + "," + self.modelStep
        #intSize = int(data.length())
        #logWriter.write(data,0,intSize)
        #logWriter.newLine()
        #data = ""
```

```
          #data = "Number Criminals: " + "," + numCrimAtNode + "," + "Number Agents at Risk: " +
numAgentAtRisk
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

        # Decide to Commit Robbery
        # Calculate SUITABILITY of victim but first check to see if there is a targetAgent found
        targetExists = false
        if targetSet == true:
          #print "Current Agent with highest wealth ", targetAgent.name
          suitability = targetWealth - crimWealth +
Random.uniform.nextIntFromTo(self.MIN_SUITABILITY, self.MAX_SUITABILITY)
          targetExists = true

        # Series of checks necessary to evaluate guardianship value calculated earlier
        # If G < 1 then there is a lack of capable guardians so commitCrime = true
        # If G = 1 then randomly assign T or F with equal probability
        # If G >= 2 then too many guardians so commitCrime = false
        #print "PreCommit Crime Guardianship is: ", guardianship
        #print "Suitability is: ", suitability

        # Check to make sure a target exists and evaluate suitability and guardianship
        if targetExists == true and suitability >= 0 and guardianship < 1 and copPresent == false:
#commit crime
          # Exchange one units of wealth
          # Subtract one unit from victim
          #print "Victim Name ", targetAgent.name
          #print "Victims current wealth ", targetAgent.wealth
          targetAgent.wealth = targetAgent.wealth - 1
          #print "Victims new wealth ", targetAgent.wealth
          #print "Offender Name: " + criminalAgent.name + "Timer value of: " +
criminalAgent.timerRepeat
          #print "Offenders current wealth ", criminalAgent.wealth

          # Add one unit of wealth to criminal
          criminalAgent.wealth = criminalAgent.wealth + 1
          #print "Offenders new wealth ", criminalAgent.wealth

          # Start the timer until citizen can offend again
          criminalAgent.timerRepeat = 1

          # Code to log the offense for that specific place
          if (currentPlace != None):
            currentPlace.totalRob = currentPlace.totalRob + 1
            #print "Robbery at Node: ", currentPlace.STRCL_
            #print "Total Robberies at Specific Node  ", currentPlace.totalRob
            # Log the offense at model level
            self.totRob = self.totRob + 1

            # Log offending and victimization for agents involved
            criminalAgent.numOffen = criminalAgent.numOffen + 1
            targetAgent.numVict = targetAgent.numVict + 1

            ##################################DEBUG
            #data = "ROBBERY:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            ##################################DEBUG
            #data = "WEALTH:  CriminalAgent: " + criminalAgent.wealth + "," + "TargetAgent: " +
targetAgent.wealth
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

            #if targetAgent.criminalPropensity == true:
               ##################################
               #data = "Model step:  " + self.modelStep
               #intSize = int(data.length())
               #logWriter.write(data,0,intSize)
               #logWriter.newLine()
               #data = ""
               #data = "Target Agent: " + targetAgent.name + "is a criminal"
               #intSize = int(data.length())
               #logWriter.write(data,0,intSize)
               #logWriter.newLine()
               #data = ""

          else:
             print "Unable to log robbery at strnode: " + occupied.strnode + " during tick: " +
self.modelStep

        # Random decision to commit robbery
        elif targetExists == true and suitability >= 0 and guardianship == 1 and copPresent == false:
          randDecision = guardianship + Random.uniform.nextIntFromTo(self.MIN_SUITABILITY,
self.MAX_SUITABILITY)
          #print "Random Decision: ", randDecision
          if randDecision == 1:
            break
          elif randDecision < 1:
            # Exchange one units of wealth
            #print "Random: Victim Name ", targetAgent.name
            #print "Random: Victims current wealth ", targetAgent.wealth
            targetAgent.wealth = targetAgent.wealth - 1
            #print "Victims new wealth ", targetAgent.wealth
            #print "Random: Offender Name " + criminalAgent.name + "Timer value of: " +
criminalAgent.timerRepeat
            #print "Random: Offenders current wealth ", criminalAgent.wealth
            criminalAgent.wealth = criminalAgent.wealth + 1
            #print "Offenders new wealth ", criminalAgent.wealth

            # Start the timer until citizen can offend again
            criminalAgent.timerRepeat = 1

            # Log the offense at the specific place
            if (currentPlace != None):
```

```
          currentPlace.totalRob = currentPlace.totalRob + 1
          #print "Robbery at Node: ", currentPlace.STRCL_
          #print "Total Robberies at Specific Node  ", currentPlace.totalRob
          # Log the offense at model level
          self.totRob = self.totRob + 1

          # Log offending and victimization for agents involved
          criminalAgent.numOffen = criminalAgent.numOffen + 1
          targetAgent.numVict = targetAgent.numVict + 1

          ###################################
          #data = "ROBBERY:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
          #intSize = int(data.length())
          #logWriter.write(data,0,intSize)
          #logWriter.newLine()
          #data = ""

          #if targetAgent.criminalPropensity == true:
            ###################################
            #data = "Target Agent: " + targetAgent.name + "is a criminal"
            #intSize = int(data.length())
            #logWriter.write(data,0,intSize)
            #logWriter.newLine()
            #data = ""

        else:
          print "Unable to log random decision robbery at strnode: " + occupied.strnode + " during tick:
" + self.modelStep

      ###################################
      #data = "FINAL:  CriminalAgent: " + criminalAgent.name + "," + "TargetAgent: " +
targetAgent.name
      #intSize = int(data.length())
      #logWriter.write(data,0,intSize)
      #logWriter.newLine()

    # Loop that logs deterrence effect of cops
    #print "Cop was present and I made it inside for loop"
    if offenderAtNode == true and copPresent == true and numAgentAtRisk >= 1 and targetWealth > 0:
      currentPlace.totPrevent = currentPlace.totPrevent + 1
      self.totDeter = self.totDeter + 1
      #print "Running total of crimes DETERRED is: ", self.totDeter

    # Log occurrence of potential crime situation - offender and victims present
    if offenderAtNode == true and numAgentAtRisk >= 1:
      self.totIntersect = self.totIntersect + 1
      #print "Running total of potential robbery situations: ", self.totIntersect

    #else:
      #print "Only one agent at node or cop at node."

  # Close the writer
  logWriter.close()
```

```
def writeOccupiedNodes():
  #print "Inside writeOccupiedNodes from model level"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Write out contents of ACTIVE NODES class
  # CREATE an output file and the buffered writer to write the activity times for each agent to a file
  currTick = int(self.getTickCount())
  #print "Current Tick: ", currTick
  outFileName = "C:/model_output"+away+"/occupiedSnapshot"+currTick+".csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames = "StreetNode,NumAgents,Agent1,Agent2,Agent3,Agent4,Agent5"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()

  # Loop through all the active nodes and write the agents at each node to a file
  for occupied as ActiveNode in self.activeNodes:
    #print "Number of agents at node ", occupied.agentList.size()
    #create a string of each data field to be written to the file
    tempNode = String.valueOf(occupied.strnode)
    numAgents = String.valueOf(occupied.agentList.size())
    temp = tempNode + "," + numAgents

    for i in range (0,occupied.agentList.size()):
      temp = temp + ","
      temp = temp + String.valueOf(occupied.agentList.get(i))

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

  #print "Agent:", citizens.getName()
  #print "Time Traveling: ", citizens.getTimeTraveling()
  #print "Time at Home: ", citizens.getTimeHome()
  #print "Time at Main: ", citizens.getTimeMain()
  #print "Time at Rec1: ", citizens.getTimeRec1()
  #print "Time at Rec2: ", citizens.getTimeRec2()

  # Close the file of activity path nodes
  txtWriter.close()


def idChangingEmploymentStatus():
  #print "ID CHANGE AGENTS loop"

  #SWITCH the employment status for 3% of agents in the model using a uniform distribution
  for i in range (30):                    #change to 30 for the full model
   index = Random.uniform.nextIntFromTo(0,999)    #change to 999 for full model
   agent = (Citizen)self.citizens.get(index)
   agent.changeEmpStatus = true
   #print "Changed agent name ", agent.name


def switchActivitySpace():
  # Switch the activity spaces of agents whose employment status changed
  # Allocate the times to remain at home, main, rec1 and rec2 and the time needed for travel
  # RN Distribution has been verified

  #print "Inside switchEmploymentStatus -- Model"

  for citizens as Citizen in self.citizens:
    # Get a new random number for each agent
    timeAtHome = Random.normal.nextInt()
    #print "Random time at home: ", totTimeAtHome

    # Get a new proportion to split time not at home
    randSplit = Random.uniform.nextDoubleFromTo(.1, .9)
    timeLeftForRec = 0
    if citizens.changeEmpStatus == true:
      #print "Index ", String.valueOf(index)
      #print "Agent Name: ", citizens.getName()
      #print "PreChange empStatus ", citizens.employmentStatus
      if citizens.employmentStatus == true:
        #print "My Values should be for unemployed"
        citizens.employmentStatus = false       #Employed becomes unemployed
        citizens.main = citizens.unempMain
        citizens.rec1 = citizens.unempRec1
        citizens.rec2 = citizens.unempRec2
        citizens.pathNodes = citizens.unempPathNodes
        citizens.changeEmpStatus = false
        #print "Main Node ", citizens.main
        #print "Unemp Node ", citizens.unempMain

        # Start of code to reallocate time at activity nodes based on length of circuit
        # Loop through the agents and find out how many nodes are in the activityPath
        totSteps = citizens.getPathNodes().size()
        #print "Total Steps: ", totSteps

        # Convert the travel time in steps to travel time in minutes
        citizens.timeTraveling = totSteps / 6
        timeAfterTravel = self.MODEL_DAY - citizens.timeTraveling

        # If loop to check for negative numbers
        while timeAfterTravel < timeAtHome:
          timeAtHome = Random.normal.nextInt()
        extra = timeAtHome - timeAfterTravel
        timeAfterTravel_Home = timeAfterTravel - timeAtHome
        citizens.timeHome = timeAtHome
        citizens.timeMain = int(timeAfterTravel_Home * randSplit)
        timeLeftForRec = timeAfterTravel_Home - citizens.timeMain
        citizens.timeRec1 = int(timeLeftForRec * randSplit)
        citizens.timeRec2 = timeLeftForRec - citizens.timeRec1
      else:
        #print "I am employed"
        citizens.employmentStatus = true        #Unemployed becomes employed
        citizens.main = citizens.empMain
        citizens.rec1 = citizens.empRec1
        citizens.rec2 = citizens.empRec2
        citizens.pathNodes = citizens.empPathNodes
        citizens.changeEmpStatus = false
        #print "Main Node ", citizens.main
        #print "Emp Node ", citizens.empMain

        # Start of code to reallocate time at activity nodes based on length of circuit
        # Loop through the agents and find out how many nodes are in the activityPath
        totSteps = citizens.getPathNodes().size()
        #print "Total Steps: ", totSteps

        # Convert the travel time in steps to travel time in minutes
        citizens.timeTraveling = totSteps / 6
        timeAfterTravel = self.MODEL_DAY - citizens.timeTraveling

        # If loop to check for negative numbers
        while timeAfterTravel < timeAtHome:
          timeAtHome = Random.normal.nextInt()

        extra = timeAtHome - timeAfterTravel
        timeAfterTravel_Home = timeAfterTravel - timeAtHome
        citizens.timeHome = timeAtHome
        citizens.timeMain = (timeAfterTravel_Home/2)
        citizens.timeRec1 = int(citizens.timeMain * randSplit)
        citizens.timeRec2 = citizens.timeMain - citizens.timeRec1

      #print "PostChange ", citizens.employmentStatus


def initCops():
  print "Inside init cops"
  #Randomly assign the cops to a starting location.

  # Use the Places to get the strnode
  for i in range (self.COPS):           #assign 2000 cop agents for testing and 200 cops for final model
    index = Random.uniform.nextIntFromTo(0, self.NUM_PLACES - 1)
    #print "Index ", index
    cop = Cop()
    cop.setModel(self)

    node = (Place)self.places.get(index)
    #print "FOUND a place " , node.STRCL_

    # Log that cop started at this node
    node.copStart = 1
    cop.setLocation(node)
    cop.setStrnode(node.STRCL_)
    self.cops.add(cop)


def resetAgentsDaily():
  #print "Inside resetAgentsDaily"

  for citizen as Citizen in self.citizens:
    citizen.atActivity = true
    citizen.atRisk = false
    citizen.moveStatus = false
    citizen.position = 0
    citizen.timeCounter = 0
    citizen.timerHome = 0
    citizen.timerMain = 0
    citizen.timerRec1 = 0
    citizen.timerRec2 = 0
    #print "Counter at reset agent: " + citizen.timerRepeat


def createCitizenTravelOutputFiles():
  print "Inside createCitizenTravelOutputFiles"

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Create an output file for model runtime statistics
  # Average number of agents unemployed, average wealth, robbery rate, exposure rate, etc.
  currTick = int(self.getTickCount())
  outFileName = "C:/model_output"+away+"/citizenChar.csv"
  dataWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Tick,NumChangeEmp,NumUnemployed,TotWealth,AvgWealth,RobRate,TotVictims,RepeatVict,Tot
Offen,RepeatOffen,NumExp,PercExposed,NumTravel,PercTraveling,numActiveOffenders,numWaitin
gOffenders,cumDeter,cumIntersect,cumRobberies"
  intSize = int(columnNames.length())
  dataWriter.write(columnNames,0,intSize)
  dataWriter.newLine()
  dataWriter.close()


def writeCitizenTravInfotoFiles():
  #print "Inside writeCitizenTravInfotoFiles"
  away = int(self.SOCIETAL_TIMEAWAY*100)

 # Loop through all citizen agents and write out the specified fields
  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    outFileName = "C:/model_output"+away+"/citizens"+agentName+".csv"
    txtWriter = BufferedWriter(FileWriter(outFileName, true))

    tempName = currTick
    home = String.valueOf(citizen.timerHome)
    temp = tempName + "," + home
    main = String.valueOf(citizen.timerMain)
    temp = temp + "," + main
    rec1 = String.valueOf(citizen.timerRec1)
    temp = temp + "," + rec1
    rec2 = String.valueOf(citizen.timerRec2)
    temp = temp + "," + rec2
    travel = String.valueOf(citizen.totTimeTraveling)
    temp = temp + "," + travel
    expose = String.valueOf(citizen.totTimeExposed)
    temp = temp + "," + expose
    vict = String.valueOf(citizen.numVict)
    temp = temp + "," + vict
    offen = String.valueOf(citizen.numOffen)
    temp = temp + "," + offen
    ahome = String.valueOf(citizen.timeHome)
    temp = temp + "," + ahome
    amain = String.valueOf(citizen.timeMain)
    temp = temp + "," + amain
    arec1 = String.valueOf(citizen.timeRec1)
    temp = temp + "," + arec1
    arec2 = String.valueOf(citizen.timeRec2)
    temp = temp + "," + arec2
    atravel = String.valueOf(citizen.timeTraveling)
    temp = temp + "," + atravel

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()
    txtWriter.close()
```

**def writeModelRunData():**

```
  print "Inside writeModelRunData"
  away = int(self.SOCIETAL_TIMEAWAY*100)
  print away

  # CREATE an output file and the buffered writer to write the activity times for each agent to a file
  modelRun = 1
  self.LOG_FILE = "C:/model_output"+away+"/RunDatav"+ modelRun + ".csv"
  txtWriter = BufferedWriter(FileWriter(self.LOG_FILE))

  # Put a model run header
  header =  "Model run: " + modelRun
  intSize = int(header.length())
  txtWriter.write(header,0,intSize)
  txtWriter.newLine()

  # Add parameter information
  seed = Random.getSeed()
  nxtLine = "Random Number Seed: " + seed
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Time to reoffend: " + self.REPEAT
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  nxtLine = "Wealth: " + "Mean " + self.WEALTH_MEAN + ", Standard Deviation " +
self.WEALTH_SD
  intSize = int(nxtLine.length())
  txtWriter.write(nxtLine,0,intSize)
  txtWriter.newLine()

  # Close text writer
  txtWriter.close()
```

```
def writeStatistics():
  # Writes out final statistics for all agents in one file to provide summary statistics
  # Aggregate time spent at home, main, rec1, rec2, travel, and exposed.
  # Assigned time to spend at home, main, rec1, rec2, travel.
  # Total number of offenses and victimizations.

  # Create a file
  away = int(self.SOCIETAL_TIMEAWAY*100)

  outFileName = "C:/model_output"+away+"/statistics.csv"
  txtWriter = BufferedWriter(FileWriter(outFileName))
  columnNames =
"Agent,timerHome,timerMain,timerRec1,timerRec2,totTimeTraveling,totTimeExposed,numVict,num
Offen,assignHome,assignMain,assignRec1,assignRec2,assignTravel,criminal,wealth"
  intSize = int(columnNames.length())
  txtWriter.write(columnNames,0,intSize)
  txtWriter.newLine()

  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    name = String.valueOf(citizen.name)
    home = String.valueOf(citizen.timerHome)
    temp = name + "," + home
    main = String.valueOf(citizen.timerMain)
    temp = temp + "," + main
    rec1 = String.valueOf(citizen.timerRec1)
    temp = temp + "," + rec1
    rec2 = String.valueOf(citizen.timerRec2)
    temp = temp + "," + rec2
    travel = String.valueOf(citizen.totTimeTraveling)
    temp = temp + "," + travel
    expose = String.valueOf(citizen.totTimeExposed)
    temp = temp + "," + expose
    vict = String.valueOf(citizen.numVict)
    temp = temp + "," + vict
    offen = String.valueOf(citizen.numOffen)
    temp = temp + "," + offen
    ahome = String.valueOf(citizen.timeHome)
    temp = temp + "," + ahome
    amain = String.valueOf(citizen.timeMain)
    temp = temp + "," + amain
    arec1 = String.valueOf(citizen.timeRec1)
    temp = temp + "," + arec1
    arec2 = String.valueOf(citizen.timeRec2)
    temp = temp + "," + arec2
    atravel = String.valueOf(citizen.timeTraveling)
    temp = temp + "," + atravel
    acriminal = String.valueOf(citizen.criminalPropensity)
    temp = temp + "," + acriminal
    awealth = String.valueOf(citizen.wealth)
    temp = temp + "," + awealth

    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()
  # Close the file
  txtWriter.close()


def dataRecorder():
  #print "DATA RECORDER T0 FILE"

  # Writes out model runtime statistics
  # Average number of agents unemployed, average wealth, robbery rate, exposure rate, etc.

  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Open the output file and the buffered writer to write the information to a file
  currTick = int(self.getTickCount())
  outFileName = "C:/model_output"+away+"/citizenChar.csv"
  dataWriter = BufferedWriter(FileWriter(outFileName, true))

  # Count number of agents to change employment status
  numChange = 0
  for citizens as Citizen in self.citizens:
    if citizens.changeEmpStatus == true:
      numChange = numChange + 1
      citizens.changeEmpStatus = false

  # Count unemployed agents
  numUnemployed = 0
  numEmployed = 0
  for agent as Citizen in self.citizens:
    if agent.employmentStatus == false:
      numUnemployed = numUnemployed + 1
    elif agent.employmentStatus == true:
      numEmployed = numEmployed + 1
    else:
      print "Employment status not assigned"
  #print "Number unemployed is: ", numUnemployed
  #print "Number employed is: ", numEmployed

  # Calculate average wealth of agents
  totWealth = 0
  for citizens as Citizen in self.citizens:
    totWealth = totWealth + citizens.wealth
  aveWealth = totWealth / self.AGENTS

  # Calculate the robbery rate
  robRate = 0
  robRate = self.totRob / self.AGENTS

  # Count number of agents victimized
  totNumVict = 0
  for citizens as Citizen in self.citizens:
    if citizens.numVict > 0:
      totNumVict = totNumVict + 1

  # Count number of repeat victims
  numRepeatVict = 0
```

```
for citizens as Citizen in self.citizens:
  if citizens.numVict > 1:
    numRepeatVict = numRepeatVict + 1

# Count number of offenders
totNumOffenders = 0
for citizens as Citizen in self.citizens:
  if citizens.numOffen > 0:
    totNumOffenders = totNumOffenders + 1

# Count number of repeat offenders
numRepeatOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.numOffen > 1:
    numRepeatOffen = numRepeatOffen + 1

# Calculate the number of citizens at risk of victimization
numExp = 0
for citizens as Citizen in self.citizens:
  if citizens.atRisk:
    numExp = numExp + 1
percExp = ((numExp / self.AGENTS) * 100)

# Calculate the number of citizens traveling
numTravel = 0
for citizens as Citizen in self.citizens:
  if citizens.atActivity == false:
    numTravel = numTravel + 1
percTravel = ((numTravel / self.AGENTS) * 100)

# Calculate the number of active offenders (able to offend)
numActiveOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.criminalPropensity and citizens.atRisk and citizens.timerRepeat == 0:
    numActiveOffen = numActiveOffen + 1

# Calculate the number of waiting offenders (not able to offend)
numWaitingOffen = 0
for citizens as Citizen in self.citizens:
  if citizens.criminalPropensity and citizens.atRisk and citizens.timerRepeat > 0:
    numWaitingOffen = numWaitingOffen + 1



# Create a string of each data field to be written to the file
temp = currTick + "," + numChange + "," + numUnemployed + "," + totWealth
temp = temp + "," + aveWealth
temp = temp + "," + robRate
temp = temp + "," + totNumVict
temp = temp + "," + numRepeatVict
temp = temp + "," + totNumOffenders
temp = temp + "," + numRepeatOffen
temp = temp + "," + numExp + ","+ percExp
temp = temp + "," + numTravel + ","+ percTravel
temp = temp + "," + numActiveOffen
temp = temp + "," + numWaitingOffen
```

```
  temp = temp + "," + self.totDeter
  temp = temp + "," + self.totIntersect
  temp = temp + "," + self.totRob
  intSize = int(temp.length())
  dataWriter.write(temp,0,intSize)
  dataWriter.newLine()

  #Close the file
  dataWriter.close()
```

**def writeFinalAgents():**

```
  print "Writing Final Agents"
  away = int(self.SOCIETAL_TIMEAWAY*100)
  baseFilePath = "C:/model_output"+away+"/"
  self.writeAgents(self.places, baseFilePath + "strnodes"+away+".shp")
```

**def writeCitizenInfoPaths():**

```
  #print "Inside writeCitizenTravInfotoFiles"
  away = int(self.SOCIETAL_TIMEAWAY*100)

  # Loop through all citizen agents and write out the specified fields
  for citizen as Citizen in self.citizens:
    # Create a string of each data field to be written to the file, two fields at a time
    agentName = String.valueOf(citizen.name)
    # Write the fields describing citizen agents
    # CREATE an output file and the buffered writer to write the activity times for each agent to a file
    currTick = int(self.getTickCount())
    #print "Current Tick: ", currTick
    outFileName = "C:/model_output"+away+"/path"+agentName+".csv"
    txtWriter = BufferedWriter(FileWriter(outFileName, true))
    node = citizen.strnode
    temp = node + "," + currTick
    intSize = int(temp.length())
    txtWriter.write(temp,0,intSize)
    txtWriter.newLine()

    txtWriter.close()
```

---

**Classes**

***Place Actions***
(none)

***Citizen Actions***

**def step():**
<u>Java imports</u>
java.lang.Object
java.lang.Double
java.lang.Number
java.lang.Integer

Code

```
 #print "INSIDE CITIZEN STEP"

 # Every citizen agent evaluates their move status, if they are moving they are added to the active
nodes
 # class and are part of the decision to commit a crime.  Then the values (atRisk, atActivity, moving,
and
 # position are set for the next turn.
 isActNodePosition = 0
 isActNode = 0

 #Obtain a random number of positions to move while traveling
 #randMoveSize = Random.normal.nextInt()
 randMoveSize = int(self.model.NORM_TRAVEL.nextDouble())

 #print "AGENT: ", self.name
 #print "POSITION: ", self.position

 #<CONDITION 1 - Start (Check to make sure there is another node in the pathNodes arrayList)
 if self.position < self.pathNodes.size()-1:
  #print "Agent: " + self.name + " is at POSITION " + self.position + " in path node array"
  #print "Position: ", self.position
  #print "Total Path Nodes in List: ", self.pathNodes.size()
  #print "ModelStep Counter: ", counter

  # Associate all agents with a current street node so we can test whether they are are atRisk
  # and/or moving.
  # Read the pathNodes arrayList and convert the number with a decimal to an integer or string.

  theStrCurrentNode = self.pathNodes.get(self.position).toString()
  #print "Node as String: ", theStrCurrentNode
  token = "."
  thePartialString = theStrCurrentNode
  #searches for decimal point
  index = theStrCurrentNode.indexOf(token)
  #print "index is :", index
  thePartialString = theStrCurrentNode.substring(0, index)
  #print "Second Node as String: ", thePartialString
  self.currentNode = Integer.valueOf(thePartialString).intValue()

  # Collect all agents who are moving or recreating are at risk and need to be logged
  # at active nodes and put them in the activeNode class.

  #<<CONDITION 2 - Start
  if self.moveStatus == true or self.atRisk == true:

   # ADD an agent to the ActiveNode class.  If there is an ActiveNode agent
   # that exists with a particular strnode value then add the name of the
   # citizen agent to the agentList (an arrayList).  If there is no ActiveNode
   # with the same value as the currentNode then add a new ActiveNode agent and
   # populate the strnode number with the currentNode and add the name of the
   # citizen agent to the agentList (an arrayList).

   #print "Current Node: ", self.currentNode
   #print "The Size is : ", self.model.activeNodes.size()
```

```
      # Test to see if this is the first ActiveNode
      nodeisEqual = false
      #<CONDITION 2.1 - Start
      if self.model.activeNodes.size() <> 0:
        for occupied as ActiveNode in self.model.activeNodes:
          if self.currentNode == occupied.strnode:
            occupied.agentList.add(self.name)
            nodeisEqual = true
      #CONDITION 2.1 - End >

      #<CONDITION 2.2 - Start
      if self.model.activeNodes.size() == 0 or nodeisEqual == false:
        newAgent = ActiveNode()
        newAgent.setModel(self.model)
        newAgent.strnode = self.currentNode
        newAgent.agentList = ArrayList()
        newAgent.agentList.add(self.name)
        self.model.activeNodes.add(newAgent)
        #print "First agent of Total Agents: ", self.model.activeNodes.size()
        #print "Inside Assignment: Strnode = ", newAgent.strnode
        #print "ArrayList Value = ", newAgent.agentList.get(0)
      #CONDITION 2.2 - End >


      #print "Current Node as Integer: ", Integer.toString(self.currentNode)
      #print "Home Node: " + self.home +  " Main Node: " + self.main + " Rec 1: " + self.rec1 + " Rec 2:
" + self.rec2
    #CONDITION 2 - End >>

    #print "Home Node: " + self.home +  " Main Node: " + self.main + " Rec 1: " + self.rec1 + " Rec 2:
" + self.rec2

    # RESET values for next turn.
    # Check to see if currentNode equal to an activity node.
    # If yes, do not move but update time that agent has been at node.  If no, move to next node.
    #<<<CONDITION 3 - Start
    if self.currentNode == self.home:
      #print "Time assigned to be at HOME: ", self.timeHome
      ##print "Time Spent at Home: ", self.timeCounter
      if self.timeCounter < self.timeHome:
        self.atActivity = true
        self.atRisk = false
        self.moveStatus = false
        #Increment the timer
        self.timeCounter = self.timeCounter + 1
        self.timerHome = self.timerHome + 1
      else:
       self.atActivity = false
       self.atRisk = true
       self.moveStatus = true
       self.timeCounter = 1
       self.position = self.position + 1    #move agent to next position in pathNode array
       ##print "Agent Leaving Home and moving to position: ", self.position
    elif self.currentNode == self.main:
      #print "Time Assigned Main ", self.timeMain
      ##print "TimeCounter for Main", self.timeCounter
```

```
  if self.timeCounter < self.timeMain:
    self.atActivity = true
    self.atRisk = false
    self.moveStatus = false
    #Increment the timer
    self.timeCounter = self.timeCounter + 1
    self.timerMain = self.timerMain + 1
  else:
    self.atActivity = false
    self.atRisk = true
    self.moveStatus = true
    self.timeCounter = 1
    self.position = self.position + 1   #move agent to next position in pathNode array
    ##print "Agent Leaving Main and moving to position: ", self.position
    self.totTimeTraveling = self.totTimeTraveling + 1
    self.totTimeExposed = self.totTimeExposed + 1
elif self.currentNode == self.rec1:
  #print "Time Assigned REC1", self.timeRec1
  ##print "TimeCounter for Rec1: ", String.valueOf(self.timeCounter)
  if self.timeCounter < self.timeRec1:
    self.atActivity = true
    self.atRisk = true                  #Agents at activities are also at risk
    self.moveStatus = false
    #Increment the timer
    self.timeCounter = self.timeCounter + 1
    self.timerRec1 = self.timerRec1 + 1
    self.totTimeExposed = self.totTimeExposed + 1
  else:
    self.atActivity = false
    self.atRisk = true
    self.moveStatus = true
    self.timeCounter = 1
    self.position = self.position + 1   #move agent to next position in pathNode array
    ##print "Agent Leaving Rec1 and moving to position: ", self.position
    self.totTimeTraveling = self.totTimeTraveling + 1
    self.totTimeExposed = self.totTimeExposed + 1
elif self.currentNode == self.rec2:
  #print "Time assigned Rec2 ", self.timeRec2
  ##print "Time spent at Rec2: ", String.valueOf(self.timeCounter)
  if self.timeCounter < self.timeRec2:
    self.atActivity = true              #Agents at activities are also at risk
    self.atRisk = true
    self.moveStatus = false
    #Increment the timer
    self.timeCounter = self.timeCounter + 1
    self.timerRec2 = self.timerRec2 + 1
    self.totTimeExposed = self.totTimeExposed + 1
  else:
    self.atActivity = false
    self.atRisk = true
    self.moveStatus = true
    self.timeCounter = 1
    self.position = self.position + 1   #move agent to next position in pathNode array
    ##print "Agent Leaving Rec2 and moving to position: ", self.position
    self.totTimeTraveling = self.totTimeTraveling + 1
    self.totTimeExposed = self.totTimeExposed + 1
```

```
   else:
     ##print "AGENT enters ELSE loop for travellers--needs to check for intervening activity node
     #print "Agent time traveling incremented to " + self.totTimeTraveling + "in else of Condition 3"

     # Check to make sure the random number is positive and nonzero
     while randMoveSize <= 0:
       #randMoveSize = Random.normal.nextInt()
       randMoveSize = int(self.model.NORM_TRAVEL.nextDouble())
     target = self.position + randMoveSize   #temporarily move agent the number of randomly
generated positions
     beginRange = self.position + 1

     #print "Randommovesize: ", randMoveSize
     #print "Target position", target

     #<CONDITION 3.1 - Start
     # Check to make sure move does not take agent beyond the number of pathNodes in list
     if target >= self.pathNodes.size()-1:
       #print "Agent at end of path"
       #print self.name + "Travelling agent RETURNED HOME at Model step: " +
self.model.modelStep
       self.currentNode = self.home
       self.position = 0
       self.timeCounter = 0
       self.atActivity = true
       self.atRisk = false
       self.moveStatus = false
       self.timerHome = self.timerHome + 1
     else:
       # Test to make sure none of the skipped nodes are activity nodes
       ##print "Main node of agent ", self.main
       ##print "Target position ", target
       for i in range(beginRange, target):
         # Get each position between the current one and the target and evaluate each
         currentPosition = i
         #print "TempTarget in loop ", tempTarget
         # Associate a position with a node
         theStrTestNode = self.pathNodes.get(i).toString()
         token = "."
         #print "theStrTestNode ", theStrTestNode
         thePartialStrTestNode = theStrTestNode
         index1 = theStrTestNode.indexOf(token)
         #print "Index 1 ", index1
         thePartialStrTestNode = theStrTestNode.substring(0, index1)
         testNode = Integer.valueOf(thePartialStrTestNode).intValue()

         # If testnode is an activity node, hold its information until all nodes
         # are checked from fartherest to nearest.
         ##print "Test node being compared: ", testNode
         ##print "Current test position ", currentPosition
         if testNode == self.home:
           #print "Node " + testNode + " is the home node " + self.home
           #print "Found home activity position at: ", currentPosition
           isActNodePosition = currentPosition
           isActNode = testNode
           break
```

```
      if testNode == self.main:
        #print "Node " + testNode + " is the main node " + self.main
        ##print "Found activity position at: ", currentPosition
        isActNodePosition = currentPosition
        isActNode = testNode
        break
      if testNode == self.rec1:
        #print "Node " + testNode + " is the rec1 node " + self.rec1
        ##print "Found activity position at: ", currentPosition
        isActNodePosition = currentPosition
        isActNode = testNode
        break
      if testNode == self.rec2:
        #print "Node " + testNode + " is the rec2 node " + self.rec2
        ##print "Found activity position at: ", currentPosition
        isActNodePosition = currentPosition
        isActNode = testNode
        break

    # Final setting of self.position and activity node status for traveling agents ONLY
    #<CONDITION 3.1.1 - START
    #print "BEFORE isActNodePosition value: ", isActNodePosition
    #print "Redirected Node before if <= 0 ", isActNode
    if isActNodePosition > 0:
      #print "AFTER isActNodePosition > 0", isActNodePosition
      self.position = isActNodePosition
      self.currentNode = isActNode
      if isActNode == self.home or isActNode == self.main:
        self.atActivity = true
        self.atRisk = false
        self.moveStatus = false
        self.timeCounter = 1
        ##print "Redirected to home or main at Position " + self.position
        if isActNode == self.home:
          self.timerHome = self.timerHome + 1
        else:
          self.timerMain = self.timerMain + 1
      elif isActNode == self.rec1 or isActNode == self.rec2:
        self.atActivity = true
        self.atRisk = true
        self.moveStatus = false
        self.timeCounter = 1
        self.totTimeExposed = self.totTimeExposed + 1
        ##print "Redirected to rec1 or rec2 at position " + self.position
        if isActNode == self.rec1:
          self.timerRec1 = self.timerRec1 + 1
        else:
          self.timerRec2 = self.timerRec2 + 1
    else:
      self.position = target
      self.totTimeTraveling = self.totTimeTraveling + 1
      ##print "Agent time traveling incremented to " + self.totTimeTraveling + " in else of Condition
3.1.1"
      self.totTimeExposed = self.totTimeExposed + 1
      ##print "MOVED TO TARGET position", target
      #print "Agent is Traveling"
      #CONDITION 3.1.1 - END>
    #CONDITION 3.1 - END>

      #print "ISACTNODE", isActNodePosition
  #print "SELF.POSITION Step", self.position
  #CONDITION 3 - END>>>

 else:
  # Reset agent to home position and reset timer to 0
  #print "RESET agent to home position"
  #print "Agent: " + self.name + "RETURNED HOME at Model step: " + self.model.modelStep
  self.position = 0
  self.timeCounter = 1
  self.atActivity = true
  self.atRisk = false
  self.moveStatus = false
  self.timerHome = self.timerHome + 1

 #CONDITION 1 - End >
```

**def assignNodeInfo(String tname, int ehome, int emain, int erec1, int erec2, ArrayList ePathNodeList, int uhome, int umain, int urec1, int urec2, ArrayList uPathNodeList):**

```
 # Assigns the variable values read from the files in initCitizens() to the fields in Citizen class
 self.name = tname
 self.home = ehome
 self.empHome = ehome
 self.empMain = emain
 self.empRec1 = erec1
 self.empRec2 = erec2
 self.empPathNodes = ePathNodeList
 self.unempHome = uhome
 self.unempMain = umain
 self.unempRec1 = urec1
 self.unempRec2 = urec2
 self.unempPathNodes = uPathNodeList
 self.currentNode = self.home
```

**def payCitizens():**

```
 #print "Inside Pay Citizens"

 # Each employed citizen gets paid at designated intervals

 if self.employmentStatus == true:
  #print "Agent Name: ",self.name
  #print "Agent Old Wealth: ", self.wealth

  self.wealth = self.wealth + 5

  #print "Agent New Wealth: ", self.wealth
```

***Active Node Actions***
(none)

***Cop Actions***

**def step():**

Java imports

```
java.lang.Object
java.lang.String
uchicago.src.sim.util.SimUtilities
java.util.Arrays
java.util.List
```

Code

```
  # Every cop moves with each model tick
  places = self.model.getPlaces()
  #print "Old node: ", self.strnode

  # Shuffle the adjacent nodes of the Place where the cop is located
  # Identify number of neighbor nodes
  numNeighs = self.location.myNeighbors.size()
  maxValue = numNeighs-1

  # Generate a value
  index = Random.uniform.nextIntFromTo(0,numNeighs -1)

  #print "Move to index: " + index

  # Verification code
  #for node in range (0, numNeighs):
    #print "Neighbor " + node + " is :" + String.valueOf(self.location.getMyNeighbors().get(node))

  # Get the new node and assign it to strnode field
  # (can't just use index because index and strnode do not correspond)
  newNode = self.location.getMyNeighbors().get(index)
  self.strnode = int(String.valueOf(newNode))

  #print "New strnode: ", self.strnode

  # Do the assignment directly of the strnode to a place
  nodeFL = Float(self.strnode)
  newLocation = (Place)self.model.placeMap.get(nodeFL)
  self.location = newLocation
  #print "New location: ", self.location.STRCL_
```

***Sequence Graph***

```
totalRobberies
return self.totRob

totalDeterred
return self.totDeter

totalIntersect
return self.totIntersect
```

## Appendix 6:  Java Code to Develop Activity Nodes

### NodeAssignment.java

```
Package AssignNodesToAgents;

/**
 * Title: Assignment
 * Description: Main program to assign nodes
 * Copyright: Copyright (c) 2005
 * @author Liz Groff
 * @version 1.0
 */

//Import classes
import java.io.*;
import java.util.*;
import java.lang.Math;

public class NodeAssignment {
  public static void main(String [] args){

                //Read the first record from the source file that has the number of nodes
                //to be allocated per blockgroup or zip
                //Create an instance of the FileInputStream class for a particular
                //file.
                FileInputStream stream = null;

                //CHANGE --Change name of input file to reflect the type of node
                try {
                            //File for homes
                  //stream = new FileInputStream
                      //("C:/Projects/Dissertation/GISData/TestData/homesperblkgrpall.csv");
                  //File for jobs
                  //stream = new FileInputStream
                      //("C:/Projects/Dissertation/GISData/TestData/empperzip.csv");
                  //File for activities
                  stream = new FileInputStream
                      ("C:/Projects/Dissertation/GISData/TestData/actperblkgrp.csv");
                }
                catch (FileNotFoundException e) {
                  e.printStackTrace(System.err);
                 System.exit(1);
                }

                //CHANGE --Create and open new file to write.  Change for different nodes.
                PrintWriter out = null;

                try {
                            //Create a file to hold the random "home" street node info
                            //String allHomes =
                            //"C:/Projects/Dissertation/GISData/TestData/allHomes.csv";

                            //Create a file to hold the random "work" nodes
                            //String allHomes =
                            //"C:/Projects/Dissertation/GISData/TestData/allJobs.csv";

                            //Create a file to hold the random "work" nodes
                            String allHomes =
                            "C:/Projects/Dissertation/GISData/TestData/allAct.csv";

                            //Create a PrintWriter based on File parameter
                            out = new PrintWriter(new FileWriter(allHomes));

                            }
                            catch (IOException exc) {
                        System.out.println(exc.toString());
                            }

                //Create an InputStreamReader
                InputStreamReader reader=new InputStreamReader(stream);

                //Create a Buffered Reader that gets data from the InputStreamReader and
                //allows it to be read by the program
                BufferedReader buffer = new BufferedReader(reader);

                //Variables
                String line;
                String templine;

                //Create line variable and read line into buffer
                try {
                  //Loop through the rows to get each blkgrpid and number of homes to be
                  //allocated per blkgrpid
                  while ((line = buffer.readLine()) != null && !line.equals("")){
                    line = line.trim();
                    templine = line;
                    int nextSpace = line.indexOf(",");
                    String blkgrpid= line.substring(0, nextSpace);
                    int homes = Integer.parseInt(templine.substring(++nextSpace).trim());
                    //System.out.println("Blockgroupid passed: " + blkgrpid + "," + homes);

                    //Call to method to extract vector of streetnodeids,x,y
                    BlkgrpNodes curBlkgrpset = collectNodes(blkgrpid);//make sure this will create a new
vector each time

                    //Add call to new method
                    System.out.println("Size of the current blkgrpset is:  "+
                    curBlkgrpset.size());
                    System.out.println("Blockgrp id of current blkgrpset is:  "+
                    blkgrpid);

                    BlkgrpNodes blkgrpRand = selectRandomNodes(curBlkgrpset,homes);
                    System.out.println(blkgrpid + ", Assigned " + blkgrpRand.size() + " homes");

                    //Write to the allHomes file
                    StrNodeLoc aStrnodeRand = null;

                    for(int i=0; i < blkgrpRand.size(); i++){
                            aStrnodeRand = blkgrpRand.get(i);
                                    out.println(aStrnodeRand.toString());
```

```
                        out.flush();
            }

            //System.exit(0);
                    }

        //close the connection to file
        out.close();
        stream.close();

        }
        catch (IOException e) {
                System.err.println("\nStack Trace Output:\n");
          e.printStackTrace(System.err);
          System.err.println("\nEnd of Stack Trace Output:\n");
            System.exit(1);
        }

        System.exit(0);
        return;
        }

//Method to loop through the data file and populate the fields
public static BlkgrpNodes collectNodes(String blkgrpid){

        //Create a file input stream and try to open the file
        FileInputStream stream2 = null;

      //CHANGE -
        try {

                //use this for blkgrps/homes and activities
          stream2 = new FileInputStream
              ("C:/Projects/Dissertation/GISData/TestData/blkgrp_stnodeall.csv");

          //use this for zip codes/jobs
          //stream2 = new FileInputStream
              //("C:/Projects/Dissertation/GISData/TestData/zip_stnodeall.csv");
         }
        catch (FileNotFoundException e) {
          e.printStackTrace(System.err);
         System.exit(1);
        }

        //Create an InputStreamReader
        InputStreamReader reader2=new InputStreamReader(stream2);

        //Create a Buffered Reader
        BufferedReader buffer2 = new BufferedReader(reader2);


        /* File format must be strnode,blkgrpid,x,y with no headers for program
         *to read properly.  File type must be csv.
         */
        //Create new object to hold StrNodeLoc object
        StrNodeLoc astrnodeLoc;
```

```
BlkgrpNodes blkgrpSet = new BlkgrpNodes();
String line2;
String templine2;
String blkgrpid1= null;
String strnode = null;
String x = null;
String y = null;

try {
  //Loop through the rows and read each one
  while ((line2 = buffer2.readLine()) != null && !line2.equals("")){
              line2 = line2.trim();
              templine2= line2;
              int nextSpace2 = line2.indexOf(",");
              strnode = line2.substring(0, nextSpace2);
              line2 = line2.substring(++nextSpace2).trim();
              nextSpace2 = line2.indexOf(",");
              blkgrpid1 = line2.substring(0,nextSpace2);
              line2 = line2.substring(++nextSpace2).trim();
              nextSpace2 = line2.indexOf(",");
              x = line2.substring(0,nextSpace2);
              y = line2.substring(++nextSpace2).trim();

              //Test for current blockgrp and loop to collect the streetnodes that are
              //associated with a blkgrpid.
              //System.out.println(blkgrpid1);
              //System.out.println(blkgrpid + "," + blkgrpid1);
              if (blkgrpid1.compareTo(blkgrpid) == 0){
                  System.out.println(blkgrpid + "," + blkgrpid1);
                        int count = 0;

                        // create the new StrNodeLoc object and populate it
                                     astrnodeLoc = new StrNodeLoc(strnode,x,y);
                  //System.out.println(astrnodeLoc.toString());

                                     // add the StrNodeLoc object to the vector of BlkgrpNodes
                                     blkgrpSet.add(astrnodeLoc);
                        }
    }
            //close the connection to file
            stream2.close();
}
catch (IOException e) {
          System.err.println("\nStack Trace Output:\n");
  e.printStackTrace(System.err);
  System.err.println("\nEnd of Stack Trace Output:\n");
    System.exit(1);
}

//System.exit(0);
return blkgrpSet;
}


/*Nested for loop to generate random numbers and then choose the
* correct number of random street nodes.
```

```
	* 1) Find number of random street nodes needed.
	* 2) Generate a random number and compare it to list of streetnodes
	* 3) Write out matched StrNodeLoc objects to a file along with the
	*    blockgroupid number
	*/

	public static BlkgrpNodes selectRandomNodes(BlkgrpNodes curBlkgrpset,
		int homes){

		Random randstrnodeValues = new Random(100); //creates random number
generator
		//Mersenne randnum = new Mersenne(1);
		StrNodeLoc aStrnodeRand = null;
		BlkgrpNodes blkgrpRand = new BlkgrpNodes();

		for (int i = 0; i < homes; i++){
			int filecount = 1;
			//Generate random index number
			int index = Math.abs(randstrnodeValues.nextInt(curBlkgrpset.size()));
			System.out.println("Random index is: " + index);
			//int index = Math.abs(randnum.genrand());
			//do while (index > curBlkgrpset.size()) {//look this up
				//index = Math.abs(randnum.genrand());

			//System.out.println("Index = " + index);
			aStrnodeRand = curBlkgrpset.get(index);
			//System.out.println("Assigned Node #" + i + ": " +
aStrnodeRand.toString());
			blkgrpRand.add(aStrnodeRand); //Put it in blkgrpRand
			//System.out.println("Size of new vector: " + blkgrpRand.size());
			}

	return blkgrpRand;
	}
}
```

## StrNodeLoc.java

```
package AssignNodesToAgents;

/**
 * Title: StrNodeLoc
 * Description: StrNodeLocs are objects that represent the streetnode
 * number and x,y of a particular streetnode.
 *
 * @author Liz Groff
 * @version 1.0
 */

 //Import classes
  import java.io.*;
  import java.util.*;
  public class StrNodeLoc {
  //Constructors -- BlkgrpNodes object
  StrNodeLoc (String strnode, String x, String y){
  this.strnode = strnode;
  this.x = x;
  this.y = y;
  }

  public String toString(){
         return strnode + " , " + x + "," + y;
  }

  //variables
  private String strnode;
  private String x;
  private String y;
}
```

## *Appendix 7: Visual Basic Code to Identify Paths Among Activity Nodes*

### frmAgentPaths.frm

```
' Copyright 1995-2005 ESRI

' All rights reserved under the copyright laws of the United States.

' You may freely redistribute and use this sample code, with or without modification.

' Disclaimer: THE SAMPLE CODE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED
' WARRANTIES, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS
' FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ESRI OR
' CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY,
' OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
' SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
' INTERRUPTION) SUSTAINED BY YOU OR A THIRD PARTY, HOWEVER CAUSED AND ON
ANY
' THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
ARISING IN ANY
' WAY OUT OF THE USE OF THIS SAMPLE CODE, EVEN IF ADVISED OF THE POSSIBILITY
OF
' SUCH DAMAGE.

' For additional information contact: Environmental Systems Research Institute, Inc.

' Attn: Contracts Dept.

' 380 New York Street

' Redlands, California, U.S.A. 92373
' Email: contracts@esri.com

Option Explicit

Private m_pNetworkDataset As INetworkDataset
Private m_pNAContext As INAContext
Private m_pNALayer As INALayer

Private Sub cmbLayer_Change()

  Dim pEnumLayer As IEnumLayer
  Dim pLayer As ILayer
  Dim pNetworkLayer As INetworkLayer

  Set pEnumLayer = MapControl1.Map.Layers
  pEnumLayer.Reset

  pEnumLayer.Reset
  Set pLayer = pEnumLayer.Next
  Do While Not pLayer Is Nothing
    If pLayer.Name = cmbLayer.Text Then
      Set pNetworkLayer = pLayer
      Set m_pNetworkDataset = pNetworkLayer.NetworkDataset
    End If
    Set pLayer = pEnumLayer.Next
  Loop

  'cleanup
  Set pEnumLayer = Nothing
  Set pLayer = Nothing
  Set pNetworkLayer = Nothing

End Sub

Private Sub cmdGo_Click()

  Dim fs As FileSystemObject
  Dim inputFolder As Folder
  Dim inputFiles As Files
  Dim agentFile As File
  Dim shapefileName As String
  Dim counter As Integer

  Dim pStopFeatureClass As IFeatureClass

  Me.MousePointer = vbHourglass

  Set fs = CreateObject("Scripting.FileSystemObject")

  'Get the files to read and create shapefiles from
  Set inputFolder = fs.GetFolder(txtInputDir.Text)
  Set inputFiles = inputFolder.Files

  'Delete temporary files if they exist
  Dim oldFiles As Files
  Dim oldFile As File
  Dim tempFolder As Folder
  Set tempFolder = fs.GetFolder(txtTempLoc.Text)
  Set oldFiles = tempFolder.Files
  For Each oldFile In oldFiles
    oldFile.Delete (True)
  Next

  'Delete the output files
  Set tempFolder = fs.GetFolder(txtOutputDir.Text)
  Set oldFiles = tempFolder.Files
  For Each oldFile In oldFiles
    oldFile.Delete True
  Next

  'Loop through each file in the input Directory
  For Each agentFile In inputFiles
     counter = counter + 1
     Label5.Caption = counter

     'Create a shapefile from the agentFile
     shapefileName = Left(agentFile.Name, Len(agentFile.Name) - 4)
     Set pStopFeatureClass = CreateShapefile(txtTempLoc.Text, shapefileName)

     'Add the points to the newly created shapefile
     AddPoints agentFile, pStopFeatureClass, True

     ' Create NAContext and NASolver
     Set m_pNAContext = CreateSolverContext(m_pNetworkDataset)

     ' Get Cost Attributes
     Dim pNetworkAttribute As INetworkAttribute
     Dim i As Long
     For i = 0 To m_pNetworkDataset.AttributeCount - 1
       Set pNetworkAttribute = m_pNetworkDataset.Attribute(i)
       If pNetworkAttribute.UsageType = esriNAUTCost Then
         cmdCostAttribute.AddItem pNetworkAttribute.Name
       End If
     Next i
     cmdCostAttribute.ListIndex = 0

     ' Load locations from FC
     LoadNANetworkLocations m_pNAContext, "Stops", pStopFeatureClass, 100

     'Create a Network Analysis Layer and add to ArcMap
     Set m_pNALayer = m_pNAContext.Solver.CreateLayer(m_pNAContext)
     Dim pLayer As ILayer
     Set pLayer = m_pNALayer
     pLayer.Name = m_pNAContext.Solver.DisplayName
     MapControl1.AddLayer pLayer

     SetSolverSettings m_pNAContext, cmdCostAttribute.Text, False, False

     ' Compute the route
     Dim strMsg As String
     Dim pGPMessages As IGPMessages
```

```
    Set pGPMessages = New GPMessages
    strMsg = Solve(m_pNAContext, pGPMessages)

  If strMsg = "OK" Then
    lstOutput.AddItem "Solve: Route length: " + Format(GetRouteOutput(m_pNAContext, "Routes"),
"######0.00") + " " + ""
  Else
    lstOutput.AddItem "Solve: " + strMsg

    ' Display  Error/Warning/Informative Messages
    If Not pGPMessages Is Nothing Then
      For i = 0 To pGPMessages.Count - 1
        Select Case pGPMessages.GetMessage(i).Type
          Case esriGPMessageTypeError
            lstOutput.AddItem "Error " & Str(pGPMessages.GetMessage(i).ErrorCode) & " " &
pGPMessages.GetMessage(i).Description
          Case esriGPMessageTypeWarning
            lstOutput.AddItem "Warning " & Str(pGPMessages.GetMessage(i).ErrorCode) &
pGPMessages.GetMessage(i).Description
          Case Else
            lstOutput.AddItem "Information " & pGPMessages.GetMessage(i).Description
        End Select
      Next i
    End If
  End If

  Call AddResultsLayer

  Dim pFC1 As IFeatureClass
  Dim pL1 As IFeatureLayer
  Set pL1 = MapControl1.Layer(0)
  Set pFC1 = pL1.FeatureClass
  Dim pFC2 As IFeatureClass
  Dim pL2 As IFeatureLayer
  Set pL2 = MapControl1.Layer(2)
  Set pFC2 = pL2.FeatureClass

  Call RelQryTabExample(pL1, pFC1, "SourceOID", pFC2, "Strcl_")

  'MapControl1.Refresh

  'routine to read the junctions and write out the nodes visited
  Call NodesVisited(pL1, txtOutputDir, shapefileName, pStopFeatureClass)
  'Remove the junctions and route layer from the map to get ready to create a new one
  MapControl1.DeleteLayer 0
  MapControl1.DeleteLayer 0
  'MapControl1.Refresh

 Next

 'cleanup
 Set fs = Nothing
 Set inputFolder = Nothing
 Set inputFiles = Nothing
 Set agentFile = Nothing
 Set pStopFeatureClass = Nothing
```

```
  Set oldFiles = Nothing
  Set oldFile = Nothing
  Set tempFolder = Nothing
  Set pStopFeatureClass = Nothing

  Me.MousePointer = vbDefault

  MsgBox "DONE"
End Sub

Private Sub dirInput_Change()

  txtInputDir.Text = dirInput.Path

End Sub

Private Sub dirOutputDir_Change()

  txtOutputDir.Text = dirOutputDir.Path

End Sub

Private Sub dirTempLoc_Change()

  txtTempLoc.Text = dirTempLoc.Path

End Sub

Private Sub Form_Load()

  Dim pEnumLayer As IEnumLayer
  Dim pLayer As ILayer

  Set pEnumLayer = MapControl1.Map.Layers
  pEnumLayer.Reset
  Set pLayer = pEnumLayer.Next
  Do While Not pLayer Is Nothing
    If TypeOf pLayer Is INetworkLayer Then
      cmbLayer.AddItem pLayer.Name
    End If
    Set pLayer = pEnumLayer.Next
  Loop

  Dim pNetworkLayer As INetworkLayer

  If cmbLayer.ListCount > 0 Then
    cmbLayer.Text = cmbLayer.List(0)
    pEnumLayer.Reset
    Set pLayer = pEnumLayer.Next
    Do While Not pLayer Is Nothing
      If pLayer.Name = cmbLayer.Text Then
        Set pNetworkLayer = pLayer
        Set m_pNetworkDataset = pNetworkLayer.NetworkDataset
      End If
      Set pLayer = pEnumLayer.Next
    Loop
  End If

  'cleanup
  Set pEnumLayer = Nothing
  Set pLayer = Nothing

End Sub

'********************************************************************************
***
' Get the Total_impedance from Route Output Class
Public Function GetRouteOutput(pContext As INAContext, strNAClass As String) As Double
    Dim intRow As Long
    Dim pTable As ITable
    Set pTable = pContext.NAClasses.ItemByName(strNAClass)
    If pTable Is Nothing Then
        GetRouteOutput = -1
        Exit Function
    End If

    Dim pCursor As ICursor
    Dim pRow As IRow
    Set pCursor = pTable.Search(Nothing, False)

    Dim pSolverSettings As INASolverSettings
    Set pSolverSettings = pContext.Solver

    Set pRow = pCursor.NextRow
    If Not pRow Is Nothing Then
        GetRouteOutput = pRow.Value(pTable.FindField("Total_" +
pSolverSettings.ImpedanceAttributeName))
    End If

    'cleanup
    Set pTable = Nothing
    Set pCursor = Nothing
    Set pRow = Nothing
    Set pSolverSettings = Nothing

End Function

Public Sub AddResultsLayer()

  Dim pFLayer As IFeatureLayer
  Dim pTraversalResultQuery As INATraversalResultQuery
  Dim pNATraversalResultEdit As INATraversalResultEdit

  Set pTraversalResultQuery = m_pNALayer.Context.Result
  Set pNATraversalResultEdit = pTraversalResultQuery

  Dim pTrackCancel As ITrackCancel

  pNATraversalResultEdit.InferGeometry "", Nothing, pTrackCancel

  Set pFLayer = New FeatureLayer
  Set pFLayer.FeatureClass = pTraversalResultQuery.FeatureClass(esriNETJunction)
  pFLayer.Name = pFLayer.FeatureClass.AliasName

  MapControl1.AddLayer pFLayer

  'MapControl1.Refresh

  'cleanup
  Set pFLayer = Nothing
  Set pTraversalResultQuery = Nothing
  Set pNATraversalResultEdit = Nothing
  Set pTrackCancel = Nothing

End Sub

Private Sub Form_Terminate()

  Set m_pNetworkDataset = Nothing
  Set m_pNAContext = Nothing
  Set m_pNALayer = Nothing

End Sub

Private Sub MapControl1_OnMouseDown(ByVal button As Long, ByVal shift As Long, ByVal x As
Long, ByVal y As Long, ByVal mapX As Double, ByVal mapY As Double)

  MapControl1.Extent = MapControl1.TrackRectangle

End Sub
```

## Basutil.bas

```
Attribute VB_Name = "basUtil"
' Copyright 1995-2005 ESRI

' All rights reserved under the copyright laws of the United States.

' You may freely redistribute and use this sample code, with or without modification.

' Disclaimer: THE SAMPLE CODE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED
' WARRANTIES, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS
' FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ESRI OR
' CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY,
' OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT
OF
' SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
' INTERRUPTION) SUSTAINED BY YOU OR A THIRD PARTY, HOWEVER CAUSED AND ON
ANY
' THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
ARISING IN ANY
' WAY OUT OF THE USE OF THIS SAMPLE CODE, EVEN IF ADVISED OF THE POSSIBILITY
OF
' SUCH DAMAGE.
' For additional information contact: Environmental Systems Research Institute, Inc.

' Attn: Contracts Dept.

' 380 New York Street

' Redlands, California, U.S.A. 92373

' Email: contracts@esri.com


Option Explicit
'******************************************************************************
'Routine CreateShapefile to create a shapefile containing the points that will be *
'used as stops on the during the route calculations                    *
'******************************************************************************
Public Function CreateShapefile(sPath As String, sName As String) As IFeatureClass ' Dont include
.shp extension

  ' Open the folder to contain the shapefile as a workspace
  Dim pFWS As IFeatureWorkspace
  Dim pWorkspaceFactory As IWorkspaceFactory
  Set pWorkspaceFactory = New ShapefileWorkspaceFactory
  Set pFWS = pWorkspaceFactory.OpenFromFile(sPath, 0)

  ' Set up a simple fields collection
  Dim pFields As IFields
  Dim pFieldsEdit As IFieldsEdit
  Set pFields = New Fields
  Set pFieldsEdit = pFields

  Dim pField As IField
  Dim pFieldEdit As IFieldEdit

  ' Make the shape field
  ' it will need a geometry definition, with a spatial reference
  Set pField = New Field
  Set pFieldEdit = pField
  pFieldEdit.Name = "Shape"
  pFieldEdit.Type = esriFieldTypeGeometry

  Dim pGeomDef As IGeometryDef
  Dim pGeomDefEdit As IGeometryDefEdit
  Set pGeomDef = New GeometryDef
  Set pGeomDefEdit = pGeomDef
  With pGeomDefEdit
    .GeometryType = esriGeometryPoint
    Set .SpatialReference = New UnknownCoordinateSystem
  End With
  Set pFieldEdit.GeometryDef = pGeomDef
  pFieldsEdit.AddField pField

  ' Add sndcl-id field
  Set pField = New Field
  Set pFieldEdit = pField
```

```
 With pFieldEdit
    .Length = 20
    .Name = "sndcl-id"
    .Type = esriFieldTypeDouble
 End With
 pFieldsEdit.AddField pField

 ' Add x field
 Set pField = New Field
 Set pFieldEdit = pField
 With pFieldEdit
    .Length = 19
    .Name = "x"
    .Type = esriFieldTypeDouble
    .Precision = 18
    .Scale = 11
 End With
 pFieldsEdit.AddField pField

 ' Add y field
 Set pField = New Field
 Set pFieldEdit = pField
 With pFieldEdit
    .Length = 19
    .Name = "y"
    .Type = esriFieldTypeDouble
    .Precision = 18
    .Scale = 11
 End With
 pFieldsEdit.AddField pField

 ' Create the shapefile
 ' (some parameters apply to geodatabase options and can be defaulted as Nothing)
 Dim pFeatClass As IFeatureClass
 Set pFeatClass = pFWS.CreateFeatureClass(sName, pFields, Nothing, _
                                  Nothing, esriFTSimple, "Shape", "")

 Set CreateShapefile = pFeatClass

 'cleanup
 Set pFWS = Nothing
 Set pWorkspaceFactory = Nothing
 Set pFields = Nothing
 Set pFieldsEdit = Nothing
 Set pField = Nothing
 Set pFieldEdit = Nothing
 Set pGeomDef = Nothing
 Set pGeomDefEdit = Nothing

End Function

'*********************************************************************************
******
'Route AddPoints which takes the rows in the textfile (csv) and puts them into the      *
'shapefile which will be used as the stops for the routes.                        *
```

```
'*****************************************************************************
******

Public Function AddPoints(textFile As File, pFeatClass As IFeatureClass, bVal As Boolean)

  Dim pFC As IFeatureCursor
  Dim pFB As IFeatureBuffer
  Dim pFieldsNew As IFields

  'Get an insert cursor and a feature buffer
  Set pFC = pFeatClass.Insert(True)
  Set pFB = pFeatClass.CreateFeatureBuffer
  Set pFieldsNew = pFB.Fields

  'Get the column index in the table for sndcl-id, x, y
  Dim indSndclid As Long
  Dim indX As Long
  Dim indY As Long
  indSndclid = pFieldsNew.FindField("sndcl-id")
  indX = pFieldsNew.FindField("X")
  indY = pFieldsNew.FindField("Y")

  'Open the textfile for reading
  Dim fs As FileSystemObject
  Set fs = CreateObject("Scripting.FileSystemObject")
  Dim theStream As TextStream
  Set theStream = fs.OpenTextFile(textFile.Path, ForReading)

  Dim theLine As String
  Dim sndclid As Double
  Dim thePosition As Integer
  Dim sx As String
  Dim sy As String
  Dim tempLine As String

  Dim pPoint As IPoint
  Dim pGeom As IGeometry

  'Read through the file and populate the shapefile with the points and id's
  Do While Not theStream.AtEndOfStream
    theLine = theStream.ReadLine
    sndclid = Val(theLine)
    thePosition = InStr(theLine, ",")
    tempLine = Right(theLine, Len(theLine) - thePosition)
    sx = Val(tempLine)
    thePosition = InStr(tempLine, ",")
    tempLine = Right(tempLine, Len(tempLine) - thePosition)
    sy = Val(tempLine)
    Set pPoint = New Point
    pPoint.x = sx
    pPoint.y = sy
    Set pGeom = pPoint
    Set pFB.Shape = pGeom

    pFB.Value(indX) = sx
    pFB.Value(indY) = sy
    pFB.Value(indSndclid) = sndclid

    pFC.InsertFeature pFB
    pFC.Flush
  Loop

  pFC.Flush

  'cleanup
  Set pFC = Nothing
  Set pFB = Nothing
  Set pFieldsNew = Nothing
  Set fs = Nothing
  Set theStream = Nothing
  Set pPoint = Nothing
  Set pGeom = Nothing

End Function

'*******************************************************************************
' Create NASolver and NAContext
'*******************************************************************************
Public Function CreateSolverContext(pNetDataset As INetworkDataset) As INAContext
    'Get the Data Element
    Dim pDENDS As IDENetworkDataset
    Set pDENDS = GetDENetworkDataset(pNetDataset)

    Dim pNASolver As INASolver
    Dim pContextEdit As INAContextEdit
    Set pNASolver = New esriNetworkAnalyst.NARouteSolver
    Set pContextEdit = pNASolver.CreateContext(pDENDS, "Route")
    pContextEdit.Bind pNetDataset, New GPMessages

    Set CreateSolverContext = pContextEdit

    'cleanup
    Set pDENDS = Nothing
    Set pNASolver = Nothing
    Set pContextEdit = Nothing

End Function

Public Sub LoadNANetworkLocations(ByRef pContext As INAContext, _
                                  ByVal strNAClassName As String, _
                                  ByVal pInputFC As IFeatureClass, _
                                  ByVal SnapTolerance As Double)

    Dim pNAClass As INAClass
    Dim pClasses As INamedSet
    Set pClasses = pContext.NAClasses
    Set pNAClass = pClasses.ItemByName(strNAClassName)

    ' delete existing Locations except if that a barriers
    pNAClass.DeleteAllRows

    ' Create a NAClassLoader and set the snap tolerance (meters unit)
    Dim pLoader As INAClassLoader
    Set pLoader = New NAClassLoader
    Set pLoader.Locator = pContext.Locator
    If SnapTolerance > 0 Then pLoader.Locator.SnapTolerance = SnapTolerance
    Set pLoader.NAClass = pNAClass

    'Create field map to automatically map fields from input class to naclass
    Dim pFieldMap As INAClassFieldMap
    Set pFieldMap = New NAClassFieldMap
    pFieldMap.CreateMapping pNAClass.ClassDefinition, pInputFC.Fields
    Set pLoader.FieldMap = pFieldMap

    'Load Network Locations
    Dim rowsIn As Long
    Dim rowsLocated As Long
    pLoader.Load pInputFC.Search(Nothing, True), Nothing, rowsIn, rowsLocated

    'cleanup
    Set pNAClass = Nothing
    Set pClasses = Nothing
    Set pLoader = Nothing
    Set pFieldMap = Nothing

End Sub

'*****************************************************************************
*****
' Get GetDENetworkDataset fom NetworkDataSet
'
*****************************************************************************
****
Public Function GetDENetworkDataset(pNetDataset As INetworkDataset) As IDENetworkDataset
    'QI from the Network Dataset to the DatasetComponent
    Dim pDSComponent As IDatasetComponent
    Set pDSComponent = pNetDataset

    'Get the Data Element
    Set GetDENetworkDataset = pDSComponent.DataElement

    'cleanup
    Set pDSComponent = Nothing

End Function
'*****************************************************************************
' Set Route Solver Settings
'*****************************************************************************
Public Sub SetSolverSettings(ByRef pContext As INAContext, _
                    ByVal sImpedanceName As String, _
                    ByVal bOneWay As Boolean, _
                    ByVal bUseHierarchy As Boolean)

    'Set Route specific Settings
    Dim pSolver As INASolver
    Set pSolver = pContext.Solver

    Dim pRteSolver As INARouteSolver
    Set pRteSolver = pSolver

    pRteSolver.OutputLines = esriNAOutputLineTrueShapeWithMeasure
    pRteSolver.CreateTraversalResult = True
    pRteSolver.UseTimeWindows = False
    pRteSolver.FindBestSequence = False
    pRteSolver.PreserveFirstStop = False
    pRteSolver.PreserveLastStop = False

    'Set generic Solver settings
    ' set the impedance attribute
    Dim pSolverSettings As INASolverSettings
    Set pSolverSettings = pSolver
    pSolverSettings.ImpedanceAttributeName = sImpedanceName

    ' Set the OneWay Restriction if necessary
    Dim restrictions As IStringArray
    Set restrictions = pSolverSettings.RestrictionAttributeNames
    restrictions.RemoveAll
    If bOneWay Then
        restrictions.Add "oneway"
    End If
    Set pSolverSettings.RestrictionAttributeNames = restrictions

    'Restrict UTurns
    pSolverSettings.RestrictUTurns = esriNFSBNoBacktrack

    ' Set the Hierachy attribute
    pSolverSettings.UseHierarchy = bUseHierarchy
    If bUseHierarchy Then
        pSolverSettings.HierarchyAttributeName = "hierarchy"
        pSolverSettings.HierarchyLevelCount = 3
        pSolverSettings.MaxValueForHierarchy(1) = 1
        pSolverSettings.NumTransitionToHierarchy(1) = 9

        pSolverSettings.MaxValueForHierarchy(2) = 2
        pSolverSettings.NumTransitionToHierarchy(2) = 9
    End If

    ' Do not forget to update the context after you set your impedance
    pSolver.UpdateContext pContext, GetDENetworkDataset(pContext.NetworkDataset), New
GPMessages

    ' Update the StreetDirectionAgent context
    Dim pNAAgent As INAAgent
    Set pNAAgent = pContext.Agents.ItemByName("StreetDirectionsAgent")
    pNAAgent.OnContextUpdated

    'cleanup
    Set pSolver = Nothing
    Set pRteSolver = Nothing
    Set pSolverSettings = Nothing
    Set restrictions = Nothing
    Set pNAAgent = Nothing

End Sub
```

```
'*******************************************************************************
' Solve the problem
'*******************************************************************************
Public Function Solve(ByVal pNAContext As INAContext, ByVal pGPMessages As IGPMessages)
As String
On Error GoTo FAIL

   'Solving the Problem
   Solve = "Error when solving"
   Dim IsPartialSolution As Boolean
   IsPartialSolution = pNAContext.Solver.Solve(pNAContext, pGPMessages, Nothing)

   If IsPartialSolution = False Then
      Solve = "OK"
   Else
      Solve = "Partial Solution"
   End If

   Exit Function
FAIL:

   If Err.Number Then
      Solve = Solve + " Error # " + Str(Err.Number) + " Description " + Err.Description
   End If
End Function

'********************************************************************************
*********
'Populate output file containing nodes visited in route for an agent                    *
'********************************************************************************
*********
Public Sub NodesVisited(ByVal pJL As IFeatureLayer, outPath As String, outFile As String,
pStopsFC As IFeatureClass)

  'Create a new shapefile to put nodes into
  Dim pOutputFeatureClass As IFeatureClass
  Set pOutputFeatureClass = CreateShapefilePaths(outPath, outFile & "path")

  'Set up file for output
  Dim pOutputFC As IFeatureCursor
  Dim pOutputFB As IFeatureBuffer
  Dim pOutputFields As IFields

  'Get an insert cursor and a feature buffer
  Set pOutputFC = pOutputFeatureClass.Insert(True)
  Set pOutputFB = pOutputFeatureClass.CreateFeatureBuffer
  Set pOutputFields = pOutputFB.Fields

  'Get the column index for the Output table for sndcl-id, x, y, step, path number,arc_
  Dim idxOutputSndclid As Long
  Dim idxarcid As Long
  Dim idxOutputX As Long
  Dim idxOutputY As Long
  Dim idxOutputPath As Long
  Dim idxOutputStep As Long
```

```
idxOutputSndclid = pOutputFields.FindField("sndcl-id")
idxarcid = pOutputFields.FindField("arc_")
idxOutputX = pOutputFields.FindField("X")
idxOutputY = pOutputFields.FindField("Y")
idxOutputPath = pOutputFields.FindField("path_num")
idxOutputStep = pOutputFields.FindField("step")

'Create a featurecursor with the records from our Original stops
Dim pStopsFeatureCursor As IFeatureCursor
Set pStopsFeatureCursor = pStopsFC.Search(Nothing, False)
Dim pStopFeature As IFeature
Set pStopFeature = pStopsFeatureCursor.NextFeature

'Get the column index for the Original stops
Dim idxStopsSndclid As Long
Dim idxStopsX As Long
Dim idxStopsY As Long
idxStopsSndclid = pStopsFC.FindField("sndcl-id")
idxStopsX = pStopsFC.FindField("x")
idxStopsY = pStopsFC.FindField("y")

'Get the first layer in the mapcontrol which should be our junctions
Dim pJunctionsLayer As ILayer
Set pJunctionsLayer = pJL
'Get the Junctions featureclass
Dim pJunctionsFeatureLayer As IFeatureLayer
Set pJunctionsFeatureLayer = pJunctionsLayer
Dim pJunctionsFeatureClass As IFeatureClass
Set pJunctionsFeatureClass = pJunctionsFeatureLayer.FeatureClass

'Create a featurecursor with the records from our junctions
Dim pJunctionFeatureCursor As IFeatureCursor
Dim pJunctionFeature As IFeature

'Get the column index for the Junctions file
Dim pTable As ITable
Dim pDisplayTable As IDisplayTable
Set pDisplayTable = pJL
Set pTable = pDisplayTable.DisplayTable
Set pJunctionFeatureCursor = pTable.Search(Nothing, False)
Set pJunctionFeature = pJunctionFeatureCursor.NextFeature

Dim idxSourceID As Long
Dim idxSourceOID As Long
Dim idxArcVal As Long
idxSourceID = pJunctionFeatureCursor.FindField("Junctions.SourceID")
idxSourceOID = pJunctionFeatureCursor.FindField("Junctions.SourceOID")
idxArcVal = pJunctionFeatureCursor.FindField("strnodes_astrnodes.Arc_")

'Other Variables needed to populate the output
Dim pPoint As IPoint
Dim pGeom As IGeometry
Dim pathnum As Integer
Dim counter As Integer
pathnum = 1
```

```
  Do While Not pJunctionFeature Is Nothing
    Set pPoint = New Point
    If pJunctionFeature.Value(idxSourceID) = 5 Then
      pOutputFB.Value(idxOutputPath) = pathnum
      pathnum = pJunctionFeature.Value(idxSourceOID)
      pOutputFB.Value(idxOutputSndclid) = pStopFeature.Value(idxStopsSndclid)
      Set pStopFeature = pStopsFeatureCursor.NextFeature
    ElseIf pJunctionFeature.Value(idxSourceID) = 2 Then
      pOutputFB.Value(idxOutputSndclid) = pJunctionFeature.Value(idxSourceOID)
      pOutputFB.Value(idxOutputPath) = pathnum
    End If
    pOutputFB.Value(idxOutputStep) = counter
    Set pPoint = pJunctionFeature.Shape
    Set pOutputFB.Shape = pJunctionFeature.Shape
    pOutputFB.Value(idxOutputX) = pPoint.x
    pOutputFB.Value(idxOutputY) = pPoint.y
    pOutputFB.Value(idxarcid) = pJunctionFeature.Value(idxArcVal)
    pOutputFC.InsertFeature pOutputFB
    pOutputFC.Flush
    Set pJunctionFeature = pJunctionFeatureCursor.NextFeature
    counter = counter + 1
  Loop

  'cleanup
  Set pOutputFeatureClass = Nothing
  Set pOutputFC = Nothing
  Set pOutputFB = Nothing
  Set pOutputFields = Nothing
  Set pStopsFeatureCursor = Nothing
  Set pStopFeature = Nothing
  Set pJunctionsLayer = Nothing
  Set pJunctionsFeatureLayer = Nothing
  Set pJunctionsFeatureClass = Nothing
  Set pJunctionFeatureCursor = Nothing
  Set pJunctionFeature = Nothing
  Set pPoint = Nothing
  Set pGeom = Nothing
  Set pTable = Nothing
  Set pDisplayTable = Nothing

End Sub

'*********************************************************************************
*********
'Create the shapefile that will contain the nodes that were visited for a particular agent *
'*********************************************************************************
*********
Public Function CreateShapefilePaths(sPath As String, sName As String) As IFeatureClass ' Dont
include .shp extension

  ' Open the folder to contain the shapefile as a workspace
  Dim pFWS As IFeatureWorkspace
  Dim pWorkspaceFactory As IWorkspaceFactory
  Set pWorkspaceFactory = New ShapefileWorkspaceFactory
  Set pFWS = pWorkspaceFactory.OpenFromFile(sPath, 0)
```

```
' Set up a simple fields collection
Dim pFields As IFields
Dim pFieldsEdit As IFieldsEdit
Set pFields = New Fields
Set pFieldsEdit = pFields

Dim pField As IField
Dim pFieldEdit As IFieldEdit

' Make the shape field
' it will need a geometry definition, with a spatial reference
Dim pSpatRef As ISpatialReference2
Set pSpatRef = frmAgentPaths.MapControl1.SpatialReference

Set pField = New Field
Set pFieldEdit = pField
pFieldEdit.Name = "Shape"
pFieldEdit.Type = esriFieldTypeGeometry

Dim pGeomDef As IGeometryDef
Dim pGeomDefEdit As IGeometryDefEdit
Set pGeomDef = New GeometryDef
Set pGeomDefEdit = pGeomDef
With pGeomDefEdit
  .GeometryType = esriGeometryPoint
  Set .SpatialReference = pSpatRef
End With
Set pFieldEdit.GeometryDef = pGeomDef
pFieldsEdit.AddField pField

' Add sndcl-id field
Set pField = New Field
Set pFieldEdit = pField
With pFieldEdit
    .Length = 20
    .Name = "sndcl-id"
    .Type = esriFieldTypeDouble
End With
pFieldsEdit.AddField pField

' Add arc_ field
Set pField = New Field
Set pFieldEdit = pField
With pFieldEdit
    .Length = 20
    .Name = "arc_"
    .Type = esriFieldTypeDouble
End With
pFieldsEdit.AddField pField

' Add step field
Set pField = New Field
Set pFieldEdit = pField
With pFieldEdit
    .Length = 4
    .Name = "step"
    .Type = esriFieldTypeInteger
  End With
  pFieldsEdit.AddField pField

  ' Add path number field
  Set pField = New Field
  Set pFieldEdit = pField
  With pFieldEdit
    .Length = 2
    .Name = "path_num"
    .Type = esriFieldTypeInteger
  End With
  pFieldsEdit.AddField pField

' Add x field
  Set pField = New Field
  Set pFieldEdit = pField
  With pFieldEdit
    .Length = 19
    .Name = "X"
    .Type = esriFieldTypeDouble
    .Precision = 18
    .Scale = 11
  End With
  pFieldsEdit.AddField pField

  ' Add y field
  Set pField = New Field
  Set pFieldEdit = pField
  With pFieldEdit
    .Length = 19
    .Name = "Y"
    .Type = esriFieldTypeDouble
    .Precision = 18
    .Scale = 11
  End With
  pFieldsEdit.AddField pField

  ' Create the shapefile
  ' (some parameters apply to geodatabase options and can be defaulted as Nothing)
  Dim pFeatClass As IFeatureClass
  Set pFeatClass = pFWS.CreateFeatureClass(sName, pFields, Nothing, _
                                Nothing, esriFTSimple, "Shape", "")

  Set CreateShapefilePaths = pFeatClass

  'cleanup
  Set pFWS = Nothing
  Set pWorkspaceFactory = Nothing
  Set pFields = Nothing
  Set pFieldsEdit = Nothing
  Set pField = Nothing
  Set pFieldEdit = Nothing
  Set pGeomDef = Nothing
  Set pGeomDefEdit = Nothing
```

```
End Function

Public Sub RelQryTabExample(ByVal pL As IFeatureLayer, ByVal pFCls As IFeatureClass,
strFClsFld As String, ByVal pFCls2 As IFeatureClass, strFCls2Fld As String)

 ' ++ Create the MemoryRelationshipClass that defines what is to be joined
 Dim pMemRelClassFact As IMemoryRelationshipClassFactory
 Set pMemRelClassFact = New MemoryRelationshipClassFactory
 Dim pRelClass As IRelationshipClass
 Set pRelClass = pMemRelClassFact.Open("Juntions_join", pFCls2, _
 strFCls2Fld, pFCls, strFClsFld, "forward", "backward", esriRelCardinalityOneToMany)

 ' ++ Perform the join
 Dim pRelQueryTableFact As IRelQueryTableFactory
 Dim pRelQueryTab As ITable
 Set pRelQueryTableFact = New RelQueryTableFactory

 'Set pRelQueryTab = pRelQueryTableFact.Open(pRelClass, True, Nothing, Nothing, "", True, True)

 Dim pDRC As IDisplayRelationshipClass
 Set pDRC = pL
 pDRC.DisplayRelationshipClass pRelClass, esriLeftInnerJoin


End Sub
```

# Bibliography

Aitken, S. C., Cutter, S. L., Foote, K. E., & Sell, J. L. (1989). Environmental Perception and Behavioral Geography. In Wilmott & Gaile (Eds.), *Geography in America* (pp. 218-238).

Akers, R. L. (2000). *Criminological Theories: Introduction, Evaluation, and Application*. Los Angeles: Roxbury Publishing Company.

Albrecht, J. (2005). A New Age for Geosimulation. *Transactions in GIS, 9*(4), 451-454.

Albrecht, J. (Forthcoming). Dynamic GIS. In J. P. Wilson & A. S. Fotheringham (Eds.), *Handbook of GIScience*.

An, L., Linderman, M., Qi, J., Shortridge, A., & Liu, J. (2005). Exploring Complexity in a Human-Environment System: An Agent-Based Spatial Model for Multidisciplinary and Multiscale Integration. *Annals of the Association of American Geographers, 95*(1), 54-79.

Axelrod, R. (Forthcoming). Advancing the Art of Simulation in the Social Sciences. In J.-P. Rennard (Ed.), *Handbook of Research on Nature Inspired Computing for Economy and Management*. Hershey, PA: Idea Group.

Axtell, R. (2000). *Why Agents? On the Varied Motivations for Agent Computing in the Social Sciences*. The Brookings Institution. Retrieved 11/5/2004, 2004, from the World Wide Web: http://www.brook.edu/es/dynamics/papers/agents/agents.pdf

Bailey, T. C., & Gatrell, A. C. (1995). *Interactive Spatial Data Analysis*. Essex: Longman Group Limited.

Bonabeau, E. (2002). *Agent-based modeling: Methods and Techniques for Simulating Human Systems.* Paper presented at the Arthur M. Sackler Colloquium of the National Academy of Sciences, Irvine, CA.

Braga, A. A. (2001). The Effects of Hot Spots Policing on Crime. *Annals of the American Academy, 578*, 104-125.

Brantingham, P., & Brantingham, P. (1981a). Introduction: The Dimensions of Crime. In P. Brantingham & P. Brantingham (Eds.), *Environmental Criminology* (pp. 7-26). Prospect Heights, IL: Waveland Press, Inc.

Brantingham, P., & Brantingham, P. (1981b). Notes on the Geometry of Crime. In P. Brantingham & P. Brantingham (Eds.), *Environmental Criminology* (pp. 27-54). Prospect Heights, IL: Waveland Press, Inc.

Brantingham, P., & Brantingham, P. (1991). Introduction to the 1991 Reissue: Notes on Environmental Criminology. In P. Brantingham & P. Brantingham (Eds.), *Environmental Criminology* (pp. 1-6). Prospect Heights: Waveland Press Inc.

Brantingham, P., & Brantingham, P. (1991 [1981]). *Environmental Criminology*. Prospect Heights, IL: Waveland Press, Inc.

Brantingham, P., & Brantingham, P. (1993). Nodes, Paths and Edges: Considerations on the Complexity of Crime and the Physical Environment. *Journal of Environmental Psychology, 13*, 3-28.

Brantingham, P. J., & Brantingham, P. L. (1978). A Theoretical Model of Crime Site Selection. In M. D. Krohn & R. L. Akers (Eds.), *Crime, Law, and Sanctions: Theoretical Perspectives* (pp. 105-118). Beverly Hills: Sage.

Brantingham, P. L., & Brantingham, P. J. (1999). Theoretical Model of Crime Hot Spot Generation. *Studies on Crime and Crime Prevention, 8*(1), 7-26.

Brantingham, P. L., & Brantingham, P. J. (2003). *Computer Simulation as a Tool for Environmental Criminologists*. Paper presented at the presented at 11th International Symposium on Environmental Criminology and Crime Analysis, University of Cincinnati.

Brantingham, P. L., & Brantingham, P. J. (2004). Computer Simulation as a Tool for Environmental Criminologists. *Security Journal, 17*(1), 21-30.

Brantingham, P. L., & Groff, E. R. (2004). *The Future of Agent-Based Simulation in Environmental Criminology*. Paper presented at the American Society of Criminology, Nashville, TN.

Brown, D. G., Riolo, R., Robinson, D. T., North, M., & Rand, W. (2005). Spatial Process and Data Models: Toward Integration of Agent-Based Models and GIS. *Journal of Geographic Systems, 7*, 25-47.

Bureau of Labor Statistics. (2003). *Metropolitan Area Employment and Unemployment: January 2003*. Bureau of Labor Statistics, United States Department of Labor. Retrieved, 2006, from the World Wide Web: www.bls.gov/news.release/archives/metro_03262003.pdf

Bursik, R. J. J., & Grasmick, H. G. (1993). *Neighborhoods and Crime: The Dimensions of Effective Community Control*. New York, NY Lexington Books.

Calthrope, P. (1993). *The Next American Metropolis: Ecology, Community and the American Dream*. New York: Princeton Architectural Press.

Capone, D. L., & Nichols, W. W. (1976). Urban Structure and Criminal Mobility. *American Behavioral Scientist, 20*, 199-213.

Carley, K. M. (1996). *Validating Computational Models*. Pittsburgh, PA: Carnegie Mellon University.

Carlstein, T., & Thrift, N. J. (1978). Afterword: Towards a Time-Space Structured Approach to Society and Environment. In T. Carlstein & D. Parkes & N. J. Thrift (Eds.), *Human Activity and Time Geography* (pp. 225-263). New York: Halsted Press.

Chaitin, G. (1990). *Information, Randomness and Incompleteness* (Second ed.). World Scientific: Singapore.

Chen, B., Cunningham, A., Ewing, R., Peralta, R., & Visser, E. (1994). Two-Dimensional Modeling of Microscale Transport and Biotransformation in Porous Media. *Numerical Methods for Partial Differential Equations, 10*(1), 65-83.

Chorley, R. J., & Haggett, P. (Eds.). (1967). *Models in Geography*. London: Methuen & Co.

Clarke, R. V. (1983). Situational Crime Prevention: Its Theoretical Basis and Practical Scope. In M. Tonry & N. Morris (Eds.), *Crime and Justice: An Annual Review of Research* (Vol. 14). Chicago: University of Chicago Press.

Clarke, R. V., & Cornish, D. B. (1985). Modeling Offender's Decisions: A Framework for Research and Policy. In M. Tonry & N. Morris (Eds.), *Crime and Justice: An Annual Review of Research, Volume 6*. Chicago: University of Chicago Press.

Clarke, R. V., & Cornish, D. B. (2001). Rational Choice. In R. Paternoster & R. Bachman (Eds.), *Explaining Criminals and Crime* (pp. 23-42). Los Angeles: Roxbury Publishing Co.

Cohen, L. E. (1981). Modeling Crime Trends: A Criminal Opportunity Perspective. *Journal of Research in Crime and Delinquency, 18*, 138-163.

Cohen, L. E., & Felson, M. (1979). Social Change and Crime Rate Trends: A Routine Activity Approach. *American Sociological Review, 44*, 588-608.

Cohen, L. E., Kluegel, J. R., & Land, K. C. (1981). Social Inequality and Predatory Criminal Victimization: An Exposition and Test of a Formal Theory. *American Sociological Review, 46*, 505-524.

Cornish, D. B., & Clarke, R. V. (1986). Introduction. In D. B. Cornish & R. V. Clarke (Eds.), *The Reasoning Criminal: Rational Choice Perspectives on Offending* (pp. 1-13). New York: Springer-Verlag.

Costanzo, C. M., Halperin, W. C., & Gale, N. (1986). Criminal Mobility and the Directional Component in Journeys to Crime. In R. M. Figlio & S. Hakim & G. F. Rengert (Eds.), *Metropolitan Crime Patterns* (pp. 73-95). Monsey, NY: Criminal Justice Press.

Cullen, F. T., & Agnew, R. (Eds.). (1999). *Criminological Theory: Past to Present*. Los Angeles, CA: Roxbury Publishing Company.

Deadman, D., & MacDonald, Z. (2004). Offenders as Victims of Crime?: An Investigation into the Relationship between Criminal Behaviour and Victimization. *Journal of the Royal Statistical Society: Series A (Statistics in Society), 167*(1), 53.

Dibble, C. (2001). *Theory in a Complex World: GeoGraph Computational Laboratories*. Unpublished PhD Dissertation, University of California Santa Barbara, Santa Barbara.

Dibble, C. (2003). Theory in a Complex World: GeoGraph Computational Laboratories. *submitted Nystrom 2003*.

Dibble, C. (2006). Computational Laboratories for Spatial Agent-Based Models. In L. Tesfatsion & K. L. Judd (Eds.), *Handbook of Computational Economics, Vol 2: Agent-Based Computational Economics* (Vol. 2). Amsterdam: Elsevier.

Dowling, D. (1999). Experimenting on Theories. *Science in Context, 12*(2), 261-273.

Duaney, A., & Plater-Zyberk, E. (1993). The Neighborhood, the District and the Corridor. In P. Katz (Ed.), *The New Urbanism: Toward an Architecture of Community*. New York: McGraw-Hill.

Eck, J. (2005). *Using Crime Pattern Simulations to Elaborate Theory*. Paper presented at the American Society of Criminology, Toronto.

Eck, J. E. (1995a). Examining Routine Activity Theory: A Review of Two Books. *Justice Quarterly, 12*(4), 783-797.

Eck, J. E. (1995b). A General Model of the Geography of Illicit Retail Marketplaces. In J. E. Eck & L. Weisburd David (Eds.), *Crime and Place* (pp. 67-93). Monsey, NY: Willow Tree Press.

Eck, J. E., & Liu, L. (2004). *Routine Activity Theory in a RA/CA Crime Simulation.* Paper presented at the American Society of Criminology, Nashville, TN.

Eck, J. E., & Weisburd, D. L. (1995). Crime Places in Crime Theory. In J. E. Eck & L. Weisburd David (Eds.), *Crime and Place* (pp. 1-33). Monsey, NY: Willow Tree Press.

Engel-Frisch, G. (1943). Some Neglected Temporal Aspects of Human Ecology. *Social Forces, 22*(1/4), 43-47.

Epstein, J. M., & Axtell, R. (1996). *Growing Artificial Societies*. Washington DC: Brookings Institution Press.

Epstein, J. M., Steinbruner, J. D., & Parker, M. T. (2001). *Modeling Civil Violence: An Agent-Based Computational Approach* (Working Paper). Washington DC: Center on Social and Economic Dynamics, Brookings Institution.

ESRI. (2003). Business Location Data. Redlands, CA: Environmental Systems Research Institute.

ESRI. (2005). ArcGIS 9.1. Redlands, CA: Environmental Systems Research Institute.

Feeney, F. (1986). Robbers as Decision-Makers. In D. B. Cornish & R. V. Clarke (Eds.), *The Reasoning Criminal: Rational Choice Perspectives on Offending* (pp. 53-71). New York: Springer-Verlag.

Felson, M. (1986a). Linking Criminal Choices, Routine Activities, Informal Control, and Criminal Outcomes. In D. B. Cornish & R. V. Clarke (Eds.), *The Reasoning Criminal: Rational Choice Perspectives on Offending* (pp. 119-128). New York: Springer-Verlag.

Felson, M. (1986b). Predicting Crime Potential at Any Point on the City Map. In R. M. Figlio & S. Hakim & G. F. Rengert (Eds.), *Metropolitan Crime Patterns* (pp. 127-136). Monsey, NY: Criminal Justice Press.

Felson, M. (1987). Routine Activities and Crime Prevention in the Developing Metropolis. *Criminology, 25*(4), 911-931.

Felson, M. (2001). The Routine Activity Approach: A Very Versatile Theory of Crime. In R. Paternoster & R. Bachman (Eds.), *Explaining Criminals and Crime* (pp. 43-46). Los Angeles: Roxbury Publishing Co.

Felson, M. (2002). *Crime in Everyday Life* (Third Edition ed.). Thousand Oaks, CA: Sage.

Gilbert, N., & Doran, J. (Eds.). (1994). *Simulating Societies: The Computer Simulation of Social Phenomena*. London: University College London Press.

Gilbert, N., & Terna, P. (1999). *How to Build and Use Agent-based Models in Social Science.* Discussion Paper. Retrieved 9-30-2003, 2003, from the World Wide Web: http://web.econ.unito.it/terna/deposito/gil_ter.pdf

Gilbert, N., & Troitzsch, K. G. (1999). *Simulation for the Social Scientist*. Buckingham: Open University Press.

Gimblett, H. R. (Ed.). (2002). *Integrating Geographic Information Systems and Agent-based Modeling Techniques for Simulating Social and Ecological Processes*. Oxford: Oxford University Press.

Gold, J. R. (1980). *An Introduction to Behavioural Geography*. New York: Oxford University Press.

Golledge, R., & Stimson, R. J. (1997). *Spatial Behavior: A Geographical Perspective*. New York: Guilford Press.

Golledge, R. G. (1978). Learning About Urban Environments. In T. Carlstein & D. Parkes & N. J. Thrift (Eds.), *Making Sense of Time* (pp. 76-98). New York: John Wiley & Sons.

Golledge, R. G. (1983). Models of Man, Points of View, and Theory in Social Science. *Geographical Analysis, 15*(1), 57-60.

Golledge, R. G., & Timmermans, H. (1990). Applications of Behavioural Research on Spatial Problems I: Cognition. *Progress in Human Geography, 14*, 57-99.

Gove, W. R., Hughes, M., & Geerken, M. (1985). Are Uniform Crime Reports a Valid Indicator of the Index Crimes? An Affirmative Answer with Minor Qualifications. *Criminology, 23*, 451-501.

Groff, E. R. (Forthcoming-a). Simulation for Theory Testing and Experimentation: An Example Using Routine Activity Theory and Street Robbery. *Journal of Quantitative Criminology*.

Groff, E. R. (Forthcoming-b). 'Situating' Simulation to Model Human Spatio-Temporal Interactions: An Example Using Crime Events. *Transactions in GIS*.

Groff, E. R. (Manuscript available from author). The Spatio-Temporal Aspects of Routine Activities and Crime.

Groff, E. R., & McEwen, T. (2005). Disaggregating the Journey to Homicide. In F. Wang (Ed.), *Geographic Information Systems and Crime Analysis* (pp. 60-83). Hershey, PA: Idea Group.

Gunderson, L., & Brown, D. (2003). *Using a Multi-Agent Model to Predict Both Physical and Cyber Crime*. Retrieved 11/12/03, 2003, from the World Wide Web: http://vijis.sys.virginia.edu/publication/SMCMultiAgent.pdf

Hägerstrand, T. (1970). What about people in regional science? *Papers of the Regional Science Association, 24*, 7-21.

Hägerstrand, T. (1973). The Domain of Human Geography. In R. J. Chorley (Ed.), *Directions in Geography* (pp. 67-87). London: Methuen.

Hägerstrand, T. (1975). Space, Time, and Human Conditions. In A. Karlqvist & L. Lundqvist & F. Snickars (Eds.), *Dynamic Allocation of Urban Space* (pp. 3-14). Farnborough: Saxon House.

Harvey, D. (1969). *Explanation in Geography*. London: Edward Arnold Publishers.

Hawley, A. H. (1950). *Human Ecology*. New York: The Ronald Press Company.

Hindelang, M. J., Gottfredson, M. R., & Garofalo, J. (1978). *Victims of Personal Crime*. Cambridge, MA: Ballinger.

Horton, F. E., & Reynolds, D. R. (1971). Action Space Differentials in Cities. In H. McConnell & D. Yaseen (Eds.), *Perspectives in Geography: Models of Spatial Interaction* (pp. 83-102). Dekab, IL: Northern Illinois University Press.

Huisman, O., & Forer, P. (1998). *Computational Agents and Urban Life Spaces: A Preliminary Realization of the Time-Geography of Student Lifestyles.* Paper presented at the GeoComputation 98, Bristol, UK.

Huisman, O., Forer, P., & Albrecht, J. (1997). *A Geometric Model of Urban Accessibility.* Paper presented at the 25th Annual Conference of the Australian Urban and Regional Information Systems Association, Christchurch, New Zealand.

Katzman, M. T. (1981). The Supply of Criminals: A Geo-Economic Examination. In S. Hakim & G. F. Rengert (Eds.), *Crime Spillover*. Beverly Hills, CA: Sage.

Kennedy, L. W., & Forde, D. R. (1990). Routine Activities and Crime: An Analysis of Victimization in Canada. *Criminology, 28*(1), 137-151.

Kerlinger, F. N., & Lee, H. B. (2000). *Foundations of Behavioral Research* (Fourth ed.). US: Wadsworth.

Kwan, M.-P. (1998). Space-time and Intregral Measures of Individual Accessibility: A Comparative Analysis Using a Point-based Framework. *Geographical Analysis, 30*, 191-216.

Kwan, M.-P., & Lee, J. (2004). Geovisualization of Human Activity Patterns Using 3D GIS: A Time-Geographic Approach. In M. F. Goodchild & D. G. Janelle (Eds.), *Spatially Integrated Social Science* (pp. 48-66). New York: Oxford University Press.

Lempert, R. (2002). *Agent-based Modeling as Organizational and Public Policy Simulators*. Paper presented at the Arthur M. Sackler Colloquium of the National Academy of Sciences, Irvine, CA.

Lenntorp, B. (1978). A Time-Geographic Simulation Model of Individual Activity Programmes. In T. Carlstein & D. Parkes & N. J. Thrift (Eds.), *Human Geography and Time Geography* (pp. 162-199). New York: Halsted Press.

Levine, N. (2005). *CrimeStat: A Spatial Statistics Program for the Analysis of Crime Incident Locations (v 3.0)*. Washington DC: Ned Levine & Associates, Houston, TX, and the National Institute of Justice.

Liu, L., Wang, X., Eck, J., & Liang, J. (2005). Simulating Crime Events and Crime Patterns in RA/CA Model. In F. Wang (Ed.), *Geographic Information Systems and Crime Analysis* (pp. 197-213). Singapore: Idea Group.

Lynch, K. (1960). *The Image of the City*. Cambridge, MA: M.I.T. Press.

Macy, M. W., & Willer, R. (2002). From Factors to Actors: Computational Sociology and Agent-Based Modeling. *Annual Review of Sociology, 28*, 143-166.

Maguire, D. J., Batty, M., & Goodchild, M. F. (Eds.). (2005). *GIS, Spatial Analysis, and Modeling*. Redlands, CA: ESRI Press.

Manson, S. M. (2001). Calibration, Verification, and Validation (Section 2.4). In D. C. Parker & T. Berger & S. M. Manson & W. J. M. (mng. ed. (Eds.), *Agent-based Models of Land-use and Land-cover change*: http://www.csiss.org/resources/maslucc/ABM-LUCC.pdf (last accessed March 14,2005).

McIver, J. P. (1981). Criminal Mobility: A Review of Empirical Studies. In S. Hakim & G. F. Rengert (Eds.), *Crime Spillover*. Crime Spillover: Sage.

Meier, R. F., Kennedy, L. W., & Sacco, V. F. (2001). Crime and the Criminal Event Perspective. In R. F. Meier & L. W. Kennedy & V. F. Sacco (Eds.), *The Process and Structure of Crime: Criminal Events and Crime Analysis* (Vol. 9, Advances in Criminological Theory, pp. 1-28). New Brunswick, NJ: Transaction Publishers.

Messner, S. F., & Blau, J. R. (1987). Routine Leisure Activities and Rates of Crime: A Macro-Level Analysis. *Social-Forces, 65*(4), 1035-1052.

Miethe, T. D., Hughes, M., & McDowall, D. (1991). Social Change and Crime Rates: An Evaluation of Alternative Theoretical Approaches. *Social Forces, 70*(1), 165-185.

Miethe, T. D., & McDowall, D. (1993). Contextual Effects in Models of Criminal Victimization. *Social Forces, 71*, 741-759.

Miethe, T. D., Stafford, M. C., & Long, J. S. (1987). Social Differentiation in Criminal Victimization: A Test of Routine Activities/Lifestyle Theories. *American Sociological Review, 52*(2), 184-194.

Miller, E. J., Hunt, J. D., Abraham, J. E., & Salvini, P. A. (2004). Microsimulating Urban Systems. *Computers, Environment and Urban Systems, 28*(2004), 9-44.

Miller, E. J., Roorda, M. J., & Carrasoc, J. A. (2005). A Tour-based Model of Travel Mode Choice. *Transportation, 32*, 399-422.

Miller, H. J. (1991). Modelling Accessibility Using Space-Time Prism Concepts Within Geographical Information Systems. *International Journal Geographical Information Systems, 5*(3), 287-301.

Miller, H. J. (2001). *What About People in Geographic Information Science?* Retrieved, 2003, from the World Wide Web: http://www.geog.utah.edu/%7Ehmiller/papers/what_about_people.pdf

Miller, H. J. (2005). A Measurement Theory for Time Geography. *Geographical Analysis, 37*, 17-45.

Mitchell, A. (1999). *The ESRI Guide to GIS Analysis* (Vol. Volume 1: Geographic Patterns & Relationships). Redlands, CA: Environmental Systems Research Institute Press.

Moss, S., & Edmonds, B. (2005). Sociology and Simulation: Statistical and Qualitative Cross-Validation. *The American Journal of Sociology, 110*(4), 1095-1131.

Nelessen, A. C. (1994). *Visions for a New American Dream: Process, Principle and an Ordinance to Plan and Design Small Communities*. Chicago: Planners.

Newton, R. R., & Rudestam, K. E. (1999). *Your Statistical Consultant: Answers to Your Data Analysis Questions*. Thousand Oaks: Sage.

North, M. J., Collier, N. T., & Vos, J. R. (2006). Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit. *ACM Transactions on Modeling and Computer Simulation, 16*(1), 1-25.

Osgood, D. W., Wilson, J. K., O'Malley, P. M., Bachman, J. G., & Johnston, L. D. (1996). Routine Activities and Individual Deviant Behavior. *American Sociological Review, 61*, 635-655.

Ostrom, T. M. (1988). Computer Simulation: The Third Symbol System. *Journal of Experimental Psychology, 24*, 381-392.

O'Sullivan, D. (2004a). Complexity Science and Human Geography. *Transactions of the Institute of British Geographers, 29*, 282-295.

O'Sullivan, D. (2004b). Too Much of the Wrong Kind of Data: Implications for the Practice of Micro-Scale Modeling. In M. F. Goodchild & D. G. Janelle (Eds.), *Spatially Integrated Social Science* (pp. 95-107). New York: Oxford University Press.

O'Sullivan, D., & Haklay, M. (2000). Agent-based Models and Individualism: Is the World Agent-Based? *Environment and Planning A, 32*(8), 1409-1425.

Parker, D. C., Berger, T., & Manson, S. M. (2001). Agent-Based Models of Land-Use / Land-Cover Change in LUCC Report Series No. 6: LUCC Focus 1 Office Anthropological Center for Training and Research on Global Environmental Change, Indiana University.

Paternoster, R. (2001). The Structure and Relevance of Theory in Criminology. In R. Paternoster & R. Bachman (Eds.), *Explaining Criminals and Crime: Essays in Contemporary Criminological Theory* (pp. 1-10). Los Angeles, CA: Roxbury Publishing Company.

Peuquet, D. J. (1994). It's About Time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems. *Annals of the Association of American Geographers, 84*(3), 441-461.

Peuquet, D. J. (2002). *Representations of Space and Time*. New York: Guilford Press.

Popper, K. R. (1965). Normal Science and Its Dangers. In I. Lakatos & A. Musgrave (Eds.), *Criticism and the Growth of Knowledge* (pp. 51-58). London: Cambridge University Press.

Pred, A. (1967). *Behavior and Location: Foundations for a Geographic and Dynamic Location Theory, Part I*. Gleerup: Lund.

Pred, A. (1996). The Choreography of Existence: Comments on Hagerstrand's Time-Geography and Its Usefulness. *Economic Geography, 53*, 207-221.

Ratcliffe, J. H. (in press). A Temporal Constraint Theory to Explain Opportunity-based Spatial Offending Patterns. *Journal of Research in Crime and Delinquency*.

Rengert, G. (1988). The Locations of Facilities and Crime. *Journal of Security of Administration, 11*(2), 12-16.

Rengert, G. F., Piquero, A. R., & Jones, P. R. (1999). Distance Decay Reexamined. *Criminology, 37*(2), 427-445.

Ropella, G. E., Railsback, S. F., & Jackson, S. K. (2002). Software Engineering Considerations for Individual-Based Models. *Natural Resource Modeling, 15*(1), 5-22.

Rountree, P. W., & Land, K. C. (1996). Burglary Victimization, Perceptions of Crime Risk, and Routine Activities: A Multilevel Analysis Across Seattle Neighborhoods. *Journal of Research in Crime and Delinquency, 33*(2), 147-180.

Sampson, R., J., & Lauritsen, J. L. (1990). Deviant Lifestyles, Proximity To Crime, and the Offender-Victim Link in Personal Violence. *Journal of Research in Crime and Delinquency, 27*(2), 110-139.

Sampson, R. J. (1993). Linking Time and Place: Dynamic Contextualism and the Future of Criminological Inquiry. *Journal of Research in Crime and Delinquency, 30*(4), 426-444.

Sampson, R. J., & Wooldredge, J. (1987). Linking Micro and Macro Dimensions of Victimization Models. *Journal of Quantitative Criminology, 3*(4), 371-393.

Schelling, T. C. (1971). Dynamic Models of Segregation. *Journal of Mathematical Sociology, 1*, 143-186.

Schultz, R. L., & Sullivan, E. M. (1972). Developments in Simulation in Social and Administrative Science. In H. Guetzkow & P. Kotler & R. L. Schultz (Eds.), *Simulation in Social and Administrative Science: Overviews and Case-Examples* (pp. 3-47). Edgewood Cliffs, NJ: Prentice-Hall.

Shannon, D. M., & Davenport, M. A. (2001). *Using SPSS to Solve Statistical Problems: A Self-Instruction Guide*. Upper Saddle River, NJ: Prentice-Hall Inc.

Sherman, L., W., Gartin, P., & Buerger, M., E. (1989). Hot Spots of Predatory Crime: Routine Activities and the Criminology of Place. *Criminology, 27*(1), 27-55.

Sherman, L., W., & Weisburd, D. (1995). General Deterrent Effects of Police Patrol in Crime `Hot Spots': A Randomized, Controlled Trial. *Justice Quarterly, 12*(4), 625-648.

Simon, H. A. (1952). A Behavioural Model of Rational Choice. *Quarterly Journal of Economics, 69*, 99-118.

Slavin, E. (1996). An Integrated, Dynamic Approach to Travel Demand Forecasting. *Transportation, 23*(3), 313-350.

SPSS for Windows. (Version Release 11.5.0)(2002). Chicago: SPSS Inc.

Tesfatsion, L. (2001). Guest Editorial Agent-Based Modeling of Evolutionary Economic Systems. *Computation, 5*(5), 437-441.

Thrift, N. J., & Bennett, R. R. (Eds.). (1978). *Towards the Dynamic Analysis of Spatial Systems*. London: Pion Limited.

Thrift, N. J., & Pred, A. (1981). Time-Geography: A New Beginning. *Progress in Human Geography, 5*(2), 277-286.

Timmermans, H., & Golledge, R. G. (1990). Applications of Behavioural Research on Spatial Problems II: Preference and Choice. *Progress in Human Geography, 14*, 311-354.

Troitzsch, K. G. (1998). Multilevel Process Modeling in the Social Sciences: Mathematical Analysis and Computer Simulation. In W. B. G. Liebrand & A. Nowak & R. Hegselmann (Eds.), *Computer Modeling of Social Processes* (pp. 20-36). London: Sage Publications.

Troitzsch, K. G. (2004). *Validating Simulation Models.* Paper presented at the 18th European Simulation Multiconference, Magdeburg, Germany.

U.S. Census Bureau (Cartographer). (2000). *Census 2000: Summary Tape File 1 (SF1)*

U.S. Census Bureau. (2002). *County Business Patterns*. U.S. Census Bureau. Retrieved, from the World Wide Web: http://censtats.census.gov/cbpnaic/cbpnaic.shtml

Visher, C. A., & Roth, J. A. (1986). Participation in Criminal Careers. In A. Blumstein & J. Cohen & J. A. Roth & C. A. Visher (Eds.), *Criminal Careers and "Career Criminals"* (Vol. I, pp. 211-291). Washington DC: National Academy Press.

Vold, G. B., Bernard, T. J., & Snipes, J. B. (2002). *Theoretical Criminology*. Oxford: Oxford University Press.

Walmsley, D. J., & Lewis, G. J. (1993). *People and Environment: Behavioral Approaches in Human Geography*. Essex: Longman Scientific & Technical.

Walsh, D. (1986). Victim Selection Procedures Among Economic Criminals: The Rational Choice Perspective. In D. B. Cornish & R. V. Clarke (Eds.), *The*

*Reasoning Criminal: Rational Choice Perspectives on Offending* (pp. 39-52). New York: Springer-Verlag.

Wang, X., Liu, L., & Eck, J. (2004). *A Spatial Dynamic Simulation of Crime Using Agent-based Modeling.* Paper presented at the Association of American Geographers, Philadelphia, PA.

Weber, J., & Kwan, M.-P. (2002). Bringing Time Back In: A study on the Influence of Travel Time Variations and Facility Opening Hours on Individual Accessibility. *The Professional Geographer, 54*, 226-240.

Weber, J., & Kwan, M.-P. (2003). Evaluating the Effects of Geographic Contexts on Individual Accessibility: A Multlevel Approach. *Urban Geography, 24*(8), 647-671.

Weisburd, D., & Green, L. (1994). Defining the Drug Market: The Case of the Jersey City DMA System. In D. L. MacKenzie & C. D. Uchida (Eds.), *Drugs and Crime: Evaluating Public Policy Initiatives*. Newbury Park, CA: Sage.

Weisburd, D., Maher, L., & Sherman, L. (1992). Contrasting Crime General and Crime Specific Theory: The Case of Hot Spots of Crime, *Advances in Criminological Theory* (Vol. 4).

Weisburd, D. L. (2002). From Criminals to Criminal Contexts: Reorienting Crime Prevention. In E. Waring & D. Weisburd (Eds.), *Crime & Social Organization* (Vol. 10, pp. 197-216). New Brunswick, NJ: Transactions Publishers.

Weisburd, D. L., Bushway, S., Lum, C., & Yang, S.-M. (2004). Trajectories of Crime at Places: A Longitudinal Study of Street Segments in the City of Seattle. *Criminology, 42*(2), 283-321.

Weisburd, D. L., Lum, C., & Yang, S.-M. (2004). *The Criminal Careers of Places: A Longitudinal Study*. Washington DC: US Department of Justice, National Institute of Justice.

Wilhite, A. (2001). *Protection and Social Order.* Paper presented at the Computational Economics and Finance Meeting, Yale University.

Williamson, D., Mclafferty, S., McGuire Philip, G., Ross, T. A., Mollenkopf, J. H., Goldsmith, V., & Quinn, S. (2001). Tools in the Spatial Analysis of Crime. In A. Hirschfield & K. Bowers (Eds.), *Mapping and Analysing Crime Data: Lessons from Research and Practice* (pp. 187-203). London: Taylor & Francis.

Xue, Y., & Brown, D. E. (2003). A Decision Model for Spatial Site Selection by Criminals: A Foundation for Law Enforcement Decision Support. *IEEE Transactions on Systems, Man, Cybernetics - Part C: Applications and Reviews, 33*(1), 78-85.

Zahn, M. A., & Jamieson, K. M. (1997). Changing Patterns of Homicide and Social Policy. *Homicide-Studies, 1*, 190-196.