DEVELOPMENT OF A SIMULATOR FOR COMPLEX SYSTEMS: A STUDENTS' LEARNING PERSPECTIVE

Sandip C. Patel, Morgan State University, MD
Ganesh Bhatt, Morgan State University, MD
Ali Emdad, Morgan State University, MD

Abstract

The art and science of simulation is getting much attention since the 1990s. This attention has opened the doors for the educational institutions to teach simulations in different courses ranging from engineering to mathematics to computer programming. However, the students at the undergraduate level find comprehending simulations difficult, especially developing and using simulators for important fields such as complex systems. This difficulty is a result of the challenge the students face in visualizing results of an experiment. In this research, we take a novel approach of showing the practical use of the simulations to the undergraduate students. By simulating a complex system using the agent-based modeling, we visually illustrate students how micro moves produce macro patterns. The paper also provides a simple view of algorithms and the programming behind the simulations. Observing the changes in the macro pattern caused by micro pattern will help students in learning how to analyze complex systems.

Keywords: simulation, complex systems, simulator development, simulator pedagogy

INTRODUCTION

With the rise of powerful computers, simulation models based on computer algorithms have become popular. For example, the simulation models show how the behaviors of a school of virtual fish--computer-generated replicas that have been trained to swim gracefully, hunt for food, and scatter at the approach of a leopard shark or how the swarms of ants, based on pheromones, find the shortest routes for their foods. These simulations are replicas of the behaviors of the "real" fish and ant colony. Several simulation models of ants' colonies have been devised to explore practical problems like the traveling salesperson's problem of finding the shortest paths to travel all the cities that he/she wants to visit.

- In this paper, we illustrate undergraduate students how simulation can be developed and used as a methodology to collect research data and testing of proposed models.
- There are different ways of running simulations. You can use Commercial-Off-The-Shelf (COTS) software such as MATLAB® or use tools such as SAS, SPSS (which uses a "modeler" which is used for providing analytics). Companies such as

Oracle and IBM offer various simulation tools.

- Most disciplines can use simulators. Each discipline has more specific tools.

The process of developing simulation includes developing simulation methodology, design of the software, and testing generated data validity. The generated data would be as good as the model used for generating the data. In developing the simulation methodology, the researcher decides phenomena to be tested and the input and output to the simulator.

Defining Simulation

Simulation tries to imitate the real-world systems and processes to study behavior of the systems. Using a model, the behavior of the system is generated and then characteristics of the real-world systems are drawn by inference. Simulation is useful in predicting the behavior of a system under different circumstances. Simulation is an appropriate tool to study interactions of a complex system or of a subsystem within a complex system (Banks, Carson, Nelson, and Nicol, 2009). Simulation can also be appropriately used for studying environmental changes, suggesting improvements in a system, observing effects of input variables, experimenting with new designs, verifying analytical solutions, determining system requirements, and learning without the cost of disruption of on-the-job training. However, simulation is not appropriate when the problem can be solved using a common sense, can be solved analytically, it is easier to perform direct experiments, costs is too much, resources or data or time is not available, or the behavior is too complex to be defined (Banks et al., 2009).

An advantage of simulation is that it mimics the real-world system before a system is implemented. Simulation provides more concrete data compared to derived data which have underlying assumptions. The behavior of the system can be analyzed without disrupting the real system. Testing of a system is easier on a simulator than the real system. Simulation can provide insight into working of the subsystem and effect of changing variables. Simulation can be used for playing what-if analysis, finding any bottleneck, and developing understanding of mechanics of the system. The disadvantages of simulation include that the model building is not an easy job. It requires experience and training. It addition to the modeling, the analysis of the results could require special training since simulation results may be tough to interpret. Rodger (2012) argues that in many instances fuzzy logic can be an aid to simulation model. Rodger (2012) provides the example of aviation safety and vehicle health maintenance.

Simulation models can be categorized as physical or mathematical, static or dynamic, deterministic or stochastic (probabilistic), and discrete or continuous. Depending on the type and nature of the project, a simulation method is selected. For example, project sot estimation can use both probabilistic as well as deterministic approach. The deterministic approach is used when data is more accurate such as that from the

history but ineffective in considering project uncertainties. The probabilistic estimation when less information is available and can help decide the degree of cost overruns. Monte Carlo simulation is one of the probabilistic techniques. Chou (2011) used Mote Carlo simulation to as a decision tool for assessing construction project's cost and uncertainties based on project managers' judgments. In this paper we simulate an agent-based problem.

CASE DETAILS

In this paper, we illustrate how we developed simulation software that was used for an exploratory study of complex social systems. In recent years, researchers have shown a great deal of interest in studying complex systems. Even though the traditional techniques have provided powerful study-tools, a band of researchers have been dissatisfied with these techniques since such techniques provide only a macro-level understanding. Understanding complex systems requires a broader aspect of knowledge that can examine the micro-level activities to understand macro-level symptoms. For example, sociologists and economists have provided several reasons of social segregation among people of different races in city, but there has been little research on understanding how exactly social segregation, a complex system, takes place. We use this case in which, we model and simulate a characteristic of complex social system that shows how social segregations in a city take place as a result of small decisions.

The underlying principle that governs a complex system is that simple rules or behaviors lead to complex systems behavior. For example, human brain is a collection of trillions of cells and about 100 billion neurons. However, by looking at one neuron or even a collection of neurons would not tell much about human consciousness or human thinking (Coveney and Fowler, 2005). The firing of millions of neurons in the brain can provide insight on the complexity of human thinking. In a way, complexity can be examined as the behavior of interacting units, irrespective of whether atoms, ants in a colony, or neurons firing in a human brain. The rise of the electronic computer provided both the key and the catalyst to our exploration of complexity.

The behavior of complex systems cannot be analyzed by examining the behavior of a single part. To fully comprehend the behavior, a global or macro-level perspective is required. The mathematical tools used in complex systems are nonlinear dynamics, graph theory, agent based modeling, time series analysis, cellular automata, network theory, genetic algorithms and information theory depending on the problems. With the rise of powerful computers, simulation models based on computer algorithms have become popular. The simulation models show how the behaviors of a school of virtual fish - computer-generated replicas - that have been trained to swim gracefully, hunt for food, and scatter at the approach of a leopard shark, or how the swarms of ants, based on pheromones, find the shortest routes for their foods.

The theory behind agent-based models is that there are some phenomena that can provide better perspectives about the behavior of the complex systems by directly modeling them on the compute rather than analyzing through mathematical equations. The reason is that computer models provide an intuitive sense of how the models at the macro level behave (Ottino, 2003). The researcher can visualize how the behavior of a model changes under different conditions and how realistically a model can replicate the "real" phenomena. According to Ottino (2003:296), "The origins of agent-based modeling can be traced to cellular automata - rows in a check board that evolve into rows in a checkerboard that evolve into the next row based on simple rules. A physical example may be the propagation of fire in a forest. Trees may be represented as occupying a fraction of the squares in a checkerboard; the rule may be that fire propagates if two trees are adjacent via the face of a square. Thus, fire propagates though faces - up, left, and right, but not diagonally. More generally, the basic building blocks may be identical or may differ in important characteristics; moreover, these characteristics may change over time, as the agents adapt to their environment and learn from their experiences. . . ."

METHODOLOGY

Building an effective simulator starts by defining the problem that needs to be studied. Once the problem is formulated, the objective of the simulator is defined in terms of what questions need to be answered. Design of the simulator is the next step. That is, abstract features of the simulator and any assumption are delineated.

Various software packages are available for simulation such as Plant Simulation (Siemens, 2014), which is used for discrete event simulation. This simulation tool creates digital models of logistic systems to explore a system's characteristics and optimize its performance. The digital models can aid in designing and planning of a production system and also help run experiments and what-if scenarios on an existing production system. Using Plant Simulation, Byrne at al. (Byrne, Cathal, Blake, and Liston, 2013) demonstrated Dell's supply of partner selection and developed new partner selection methodology. The authors used discrete event simulation for modeling. A software for the industry such as construction can take advantage of Interactive Construction Decision Making Aid (ICDMA) (Anderson, Mukherjee, and Onder, 2009; Rojas and Mukherjee, 2006) which a simulation software for multiple simulations for studying different management strategies and conditions of a construction project. Using the simulated processes of ICDMA, the decision makers can learn about effects of allocating and re-allocating resources on project contingencies and find the best way to handle the contingencies. Tang et al. (Tang, Cass, and Mukherjee, 2013) set up a highway construction project using ICDMA. The authors studied the effects of construction management strategies on greenhouse gas emitted during the construction project. The research investigated ways in which the emission can be controlled by appropriately managing disruptive events. A simulation application can be developed using software such as MATLAB®. For example, Uzzafer (2013) presented a MATLAB® based application to computerized

parts of the simulation model he proposed. The simulation model was proposed for strategic management process of software development project which used specific risk management, cost estimation models, and project estimation tools. The goal was to aid software practitioners and academics in utilizing the system for software development projects and further research. The models simulated decision factors such as cost, risk, budget, and schedule.

The steps in simulating a problem include the following steps: (1) Problem specification and requirement analysis. During this step, the objective of the study or the project is set. Next, (2) design the simulator, (3) build the simulation model by activity such as wring code, (4) test the validity of the data collected using the simulator, and finally, (5) design experimental runs and execute the runs. In the following sections, we describe how each of these steps was conducted for the case. In the following section, we describe the case that was simulated and motivation for the development.

Simulator Details

The simulator aims at exploring the macro-level details of complex social systems, such as a city block of houses occupied by different races. More specifically, the simulator provides data to examine how individual decision of moving into a neighborhood or house affects a bigger picture of segregated neighborhoods. "The criterion is that the native wants to live on the houses that are the nearest to other houses occupied by the natives" (Schelling, 1978). To test how simple preferences of agents (individuals) can lead to complex outcomes, we simulated and executed test-runs. We created the simulator in Visual Basic.NET 2010. The program contains the GUI representing a grid of 21x21 checkboxes, with each checkbox representing a house (see Figure 1) with a total of 441 houses. The program provides features to randomly or specifically select houses as those occupied by a race (e.g., a native group).

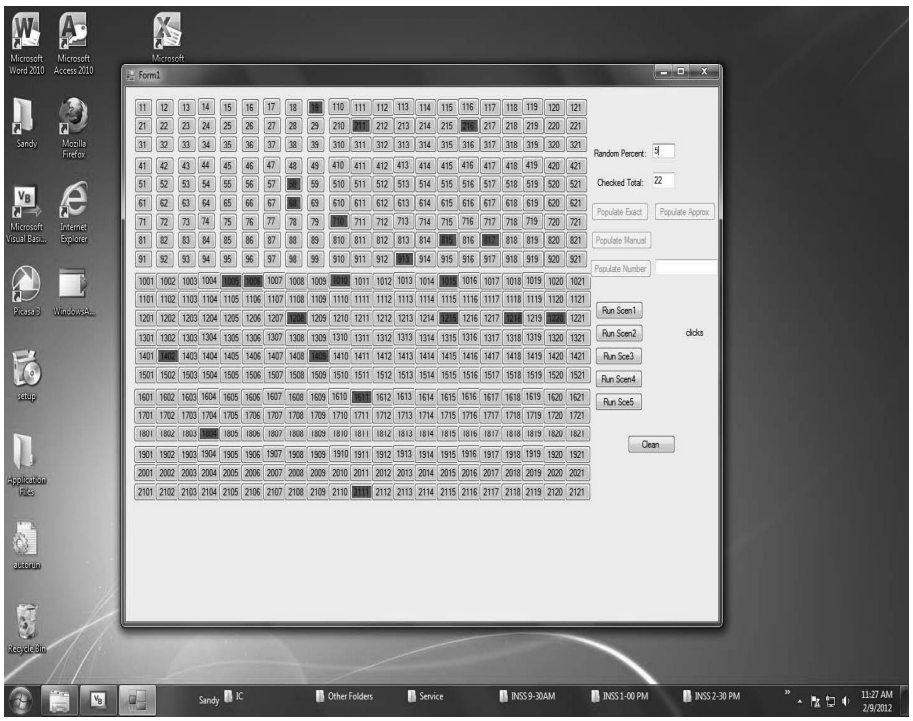
The first step in running the simulator involves marking the houses that are occupied by a racial group. We refer to this racial group as the native group. The landscape of the houses marked by the initial occupancy is referred as the initial condition. To set the initial condition, our simulator allows specifying the exact percent of the houses occupied by the race randomly, or specifying the probability of each house to be occupied by the race. All the marked houses in each of the simulation run are shown as occupied by a family of the native group, while unmarked houses are occupied by a different race (i.e., white families). Then, we run the simulator with several cycles with one specific criterion.

Simulated Cycles

As explained above, the houses marked are defined as those occupied by the "native group" in a city population. Once the initial condition is generated, we run a desired

number of cycles for showing the movement of families in and out of the city population. A cycle is meant to simulate a time period, such as a year. That is, a cycle represents the change in the neighborhood landscape of the city population within one time-unit. In other words, a cycle represents a next phase of occupancy by the native group in a given time period, such as a year. For this paper, we ran three cycles for different initial condition of the native group.

FIGURE 1: Initial Landscape: 5% Random-Occupancy by The Native Population (Green Cells Show The Native Group)



Simulated Criterion

In this study, we select a simple criterion of household movement from and to the city population. The criterion is that the native wants to live on the houses that are the nearest to other houses occupied by the natives (Schelling, 1978). As the natives attempt to move into the houses that are adjacent to other natives, white race begins to move away from the city population. Each cycle of simulation shows how the household movement changes the landscape of the neighborhood in the city population with more natives moving in adjacent to the other native. For example, if the 3rd house in the second row of the neighborhood has a native group in the second

cycle, the 2nd and the 4th houses in the third row become marked (i.e., occupied by the native group) in the third cycle. If there are more than one adjacent house occupied by the native group, then in the following cycle, each side of the consequent native houses is also marked as occupied by the natives. For example, if the 4th, 5th, and the 6th houses in row five are occupied by natives, then in the next cycle, the 3rd and the 7th houses in row five also gets occupied by the natives.

The program depicts each cycle showing what the next new occupancy would look like between natives and whites, based on the above criterion. Each of the simulation test-run displays the racial landscape of the city population. Using the criterion listed above, we ran the simulator with 5%, 10%, 20%, and 25%initial condition in which natives randomly occupy the specified percent of the houses in the city population. In each of the test run, we see an emerging pattern that clearly shows the movement of whites away from the city population and movement of natives in the city population.

SIMULATOR-DEVELOPMENT PROCESS

In this section, we provide the details of the development process of the simulator described above. The development process includes derivation of specifications and requirements, designing of the simulator software, coding, and finally, executing the software and collecting results.

Specification and Requirements

In this section, we discuss how the specifications and requirements were developed for this simulator. Hence, we start with the objective of the study and then explore what needs to be included in this simulator. As discussed above, the objective of our study was to study the complex system of social segregation. More specifically, we were interested in studying how micro decisions by an individual affect the macro feature. For studying the complex case of social segregation, we required that the simulator has certain general and specific features. Based on our goal of exactly what we wanted to study and the literature review of social segregation and simulation we delineated the following specifications. The specifications also considered the best practices in programming.

- Visual component: It was important for our study to see how individuals moved and what effect individual moving had on layout of residential areas. In other word, we wanted to be able to see both the micro movement of each individual and the macro picture. The visual component should reflect a real-life scenario as closely as possible.
- Different and random initial sample sizes: The simulator should accommodate random and different number of initial population so we can study the effect of varying initial conditions.

- State-by-state runs: The simulator should give the data about each state and its progression as we run the experiment and not just at the end.
- Types of models: Our study needed to accommodate different social behaviors. For example, choosing the neighbor on the left-side of an empty house or the right side of the empty house, or both, and so forth.
- Number of races: We needed our study to be flexible enough so that we can incorporate several races in our simulator.
- Large total number of individuals: Our study required that we had a sufficiently large number of individuals so that we can observe the patterns of segregation.
- Simplicity: We wanted our model to be simple enough so that the associated code is not too complicated to be understood, modified, or enhanced by the people other than those who developed it.
- Flexibility: We wanted our model to be flexible so that different situations can be simulated.
- User-friendly and easy interface: The simulator should have an easy and comfortable look and feel.
- Portability: The simulator should be able to run on most computer and operating systems.
- Scalability: We wanted the simulator to be scalable so that we can add different functionalities or enhance the simulator for different scenarios, for example, to examine the spread of virus.

Simulation Design

During the design stage, functions and operations of the software are described. Such descriptions include screen layouts, business rules, and process description and diagrams. The output of this stage describes the new system as a collection of modules or subsystems. After considering the specifications listed above, the different types of simulations techniques, and best practices in programming, we found that the agent-based simulation would be most appropriate in meeting the specified needs.

We decided to treat the individuals (people) as intelligent agents. The individuals' behaviors are modeled in the simulator. The behavior protocol is the decision of an individual to move in a house. The agent knowledge includes what houses are occupied. The agent attribute includes an agent's race. The environment model includes the rows of houses as environment objects. The environment state is the current state of house occupancy. The behaviors of agents depend on the agent behavior protocol and agent knowledge of the environment. The agent behaviors lead to a new state. In turn, the agents acquire the new environment state information and based on the agents' behavior, the next stage is created.

A layout of 441 houses was selected giving a sufficiently large number and realistic real-life representation with 21x21 objects each simulating a house. Based on the above requirements, we chose the design feature of the simulator that would fulfill each requirement. We decided to build an agent-based simulator. Thomas Schelling was the first to reason about the phenomenon of complex systems in social segregation (Schelling, 1978). Even though individuals of similar color might mildly prefer the company of each other, without realizing that its overall effect at the macro level. In a sense, each individual makes their own choices at the local level, but the overall effect of these choices at macro level can lead to social segregation. As Schelling argues that for example, blacks and whites may get along with one another, but a slight preference of whites to live near the whites and blacks near the blacks can over time lead to segregation. Initially, blacks and whites may be distributed randomly within the city. But some whites will move to the places where other whites are living, which are likely to increase the number of blacks in their own neighborhoods, resulting other whites to leave. Simulations can depict these phenomena how individual preferences can lead to different clusters of populations. For example, even a weak preference of living by of the same colors can lead to an overwhelming number of people around. Specific details of the design features, which correspond with the requirements, are as following.

- Visual component: We selected a screen with 21x21 houses that show the house occupancy in green at each run state.
- Different and random initial sample sizes: For effective randomness, we ran the random number generator twice. That is, using the first random number as the seed for generating the second number so that we can be sure of true randomness. Also, for the initial layout, we houses occupied by a particular race were randomly selected. There were two different methods used for randomly marking the house which are explained in the programming details.
- State-by-state runs: We used the concurrent simulation so that the visual representations are displayed when the simulator is run. Also, state modeling was selected rather than continuous.
- Types of models: We implemented only one model where both the adjacent houses (the house on the left and the house on right) of an occupied house get occupied at every state. For simplicity, we did not implement other models at this time.
- Number of races: We kept the simulator flexible enough so that other races could be added in the future but not at this time.
- Large total number of individuals: We selected 441 houses, which we considered as a large number for this study.
- Simplicity: Since the code needed to be simple, we chose the deterministic model because the probabilistic model would have been more complex. Also, we decide that the model would include only the visual component and not any statistics data. Only one agent attribute, namely the agent's race, would be considered. Other attributes such as gender, age, income group and so forth are not considered as a factor that would affect an agent's behavior.

- Flexibility: We modularized the software so that it can model different scenarios. We anticipated that we would add and enhance the model at later time. A number of routines and procedures were designed maximizing the program modularity. We chose Visual Basic.NET since it its object-oriented nature would enable us to write programs using reusable, modular objects. Also, we created the objects such as a house so that such objects can take different attributes such as a color to show if the house was occupied. Finally, we kept provisions in the simulator should be able to simulate different cases so that the future. For example, we chose to change the color of the checkbox object to show occupancy which would give multiple choices as opposed to showing the occupancy by checking on or off the checkbox which would have given only two choices.
- User-friendly and easy interface: We predominantly used GUI (graphical user interface) to depict the house occupancy landscape. In addition, we show dynamic graphics on screen for each state. Thirdly, we set the default values of the attributes and values that were needed to be supplied by a user to assist the user in deciding.
- Portability: For portability, we chose VB.NET which supports creating an executable file which that run on most computers.
- Scalability: In our simulator, we included a function that gives the user flexibility to choose a set of houses for initial occupancy rather than using the random functionality. This function would later be used for enhancing the simulator for another case such as simulating contagion.

The following lists the data structures, screen objects, and sub routines designed.

Data Structures

- Checkboxes: This data structure is a two dimensional array of objects and is used to represent the 21x21 houses. The objects of this array are checkbox controls of Visual Basic.NET. The array uses variables for defining its size so that just by changing the variable, the array size can be changed later. This feature was inserted because the design required flexibility.
- houses: This is a two dimensional integer array. It is used to keep track of house status, which can be changed to 0 and 1 to indicate the background color of the associated checkbox. The value zero is used for an empty house indicated by the light gray color of the checkbox and value 1 for the green color of the checkbox. In addition to 0 and 1, any other integer can be used in the future to accommodate more races (represented by colors) in the future. This feature accommodated the design requirement of flexibility.
- duplicate: This two dimensional integer array is used to temporarily store the values of the houses array.
- Random number: A variable of class *Random()* with its *Next()* method is used for creating random numbers.

Screen Objects

- btnRunSce1_click: This sub procedure executes designed model of the case. That is, if the house is occupied by the green race, this sub marks the house on the left and right. This sub calls fill_checkboxes(), Run_the_main_progSce1, associate(), and count_adjacent() subs.
- BtnClan_Click(): This button is used for clearing all other objects and data structures so that a new scenario can be run.

Sub Routines

- fill_checkboxes: This sub fills the checkboxes array with appropriate checkbox objects. For example, the checkboxes(1,1) element is filled with the checkbox located in the first column of the first row.
- Run_the_main_progSce1() which calls checkneighbour_Sce1 which does the actual marking of the house.
- Associate(): This sub is used for changing the background color of the houses.

When the user runs the software, the program initializes the array whose elements are checkboxes. The user chooses how s/he wants to populate the initial layout by choosing according GUI control on screen. The user has three choices: populate-exact, populate-approximately, and populate-manual.

In the following sections, we explain the process flow. The processes were divided into two parts. The following sections list the design for part 1 followed by part 2.

Process: Part 1

Process for populate-exact: The program reads the random percent of houses the user has chosen to populate. This percent is converted into total number of houses to be populated. The program goes into a loop that ends when the loop finishes marking the chosen number of houses ("total_houses"). The program within the loop first checks if the number of marked houses (marked_houses") equals the total_houses. If the loop has not completed marking all the houses, a random row and a column are selected. The house located at the intersection of this column and the row is marked.

Process for populate-approximately: The program reads the random percent of houses the user has chosen to populate. Program considers each house at a time, starting with the first house. A random number is generated between 1 and 100. If this number falls between 1 and the percent value, the house is marked. This process is repeated for all the houses on GUI.

Process for populate-manual: The program read a row number and a column number entered by the user till s/he enters "999" for the row and "999" for the column as the

termination values. The program marks each houses at a time located at the intersection of the row and the column.

Process: Part 2

The program reads the scenario that the user selects to run. Currently, a deterministic and a probabilistic models have been implemented. However, the specifications required that the software be open to include other models in the future. The data structure "house_array" is copied onto a data structure "duplicate_array", both of which are arrays. The program checks if a house is occupied. Then, it marks the houses on the left and the right of the occupied house. All these houses are marked in the duplicate_array. In order to do this marking, the program start with the first house repeats the process till it reached the last house. Once the making is completed, the program copies duplicate_arran onto house_array. The user can select to run the next cycle.

Coding for the Simulation Model

In addition to fulfilling the requirement of portability, VB.NET provides important features such as Intellisense and interactive debugging. VB.NET inherits development environment tools and services of Visual Studio and the .NET Framework.

We designed two methods for marking the initial population. For both of the methods of marking, the user uses the same textbox and enters the number in terms of percent s/he desires the houses to be marked green. Using the processes above, pseudocode was developed for the entire program. For brevity, we are showing only a part of the pseudocode below.

(a) Pseudocode for populating randomly but the exact percent of houses supplied by the user. In this method of populating, the user supplies the percent of houses s/he wants to mark green (initially occupied houses). We designed and used the following pseudocode:

- Convert the percent supplied by the use into the number of houses.
- Loop: Check the number of marked houses.
- Until that number of houses are marked:
    - Select a random row
    - Select a random column
    - Mark that house as occupied by the green race.
- Loop end.

(b) Pesudocode for populating houses with the user-supplied probability. In this method of marking initially occupied houses, each house has the user-supplied probability of getting marked. This method will give the randomly chosen houses

and yield approximately the percent of houses equal to the percent probability supplied.

- for each of the houses:
    - Flip a die (i. e., generate a random number) which has numbers 1 to 100.
    - If the die shows a number between 1 and the probability supplied by the user, mark the house.

The example of the above psuedocode is as following: If the user wants the 34% probability of house selection, a die is flipped to generate a number from 1 to 100. If the number generated is between 1 and 34, the first house gets marked. Then, this process is repeated for each of the rest of the houses.

Simulation Model Validation

We conducted experimental runs of the simulator to verify if the data produced was valid. We compared the produced data and statistics by manually counting at each step. We observed the change of agent environment to see if the simulator correctly processed the step-by-step outcome. For the randomness, we ran the test a large number of times to verify that there was not pattern in the initial selection of the house and each house was truly randomly selected when the trial was run for a large number of times.

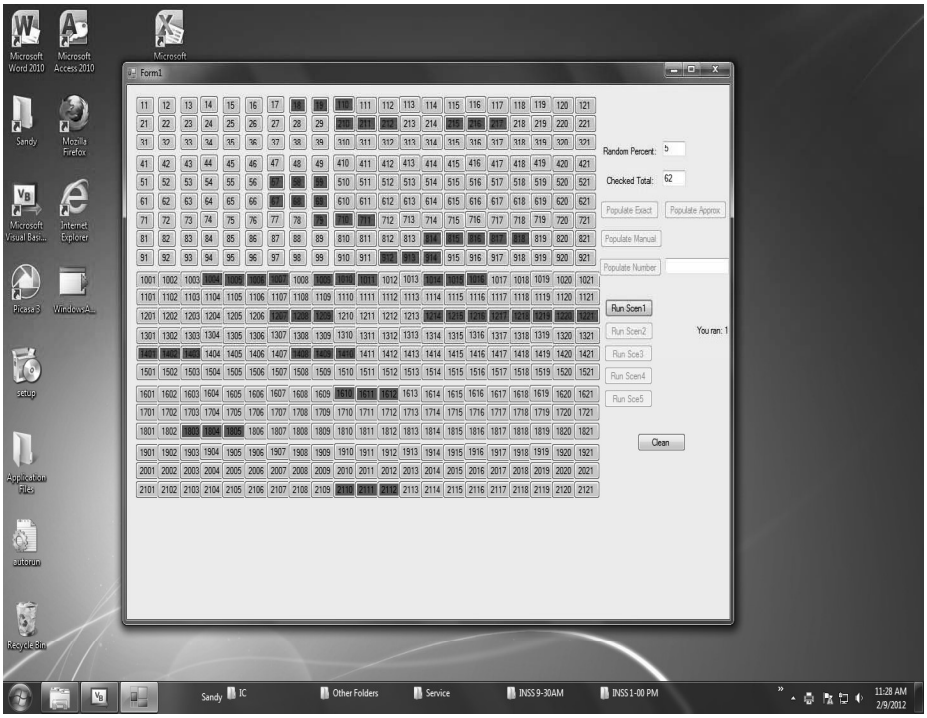
Simulator Execution and Results

To test how simple preferences can lead to complex outcome, we ran different sets of simulations by setting the initial condition of the native group to 5%, 10%, 20%, and 25% randomly. Each of the simulation was run for three cycles, where each cycle successively changes the landscape of the native group.

To illustrate how local criterion of native group can lead to a complex structure of population based on the racial preferences, we begin with a random population of native group with 5%. That mean, the initial state starts with 5% of native group in the population of 441 houses. Therefore, we begin with random 5% (i.e., 22 houses) as the initial condition of the natives and rest 419 houses as occupied by whites in the population. In this landscape, we see that all the houses, except one block of 2 houses, adjoining to each other, are occupied by single native group (see Figure 1). After the first cycle of simulation, as show in Figure 2, we find a large shift in the proportion of populations of natives in the city population. We find 15 different blocks of 3 houses adjoining to each other; 1 block with 4 houses adjoining to each other; 1 block of 5 houses adjoining to each other, and 1 block of 8 houses adjoining to each other have been occupied by native group. After running the second simulated cycle, we find a big shift in the population of the natives that varies from 1 block of 4 houses, 10 different blocks of 5 houses, 1 block of 7 houses, 1 block of 9 houses, 1 block of 10 houses, and 1 block 15 houses adjacent to each other are occupied by natives. In the 3rd cycle, the shift has been dramatic where we find

several houses consisting of 8 blocks with 7 houses adjoining to each other; 1 block of 9 houses adjoining to each other; 2 blocks of 12 houses adjoining to each other, and 2 blocks of 17 houses adjoining to each other are populated by the native group (see Figure 3). Figure 3 demonstrates how the population of native group has aggregated close to each other in repeated runs of the simulation cycles as well as how whites have successively moved away from the neighborhood.

FIGURE 2: Landscape After the First Cycle of 5% Initial Occupancy by the Native Population (Green Cells Show the Native Group)



In the other detailed example, we examine the initial condition representing 1/5th of the city population consisting of natives. We begin with random 20% (i.e., 88 houses) as the initial condition of the natives in the city. Initially, we find that 65 single houses of natives are adjoining to the white's houses; 7 different blocks of houses with 2 houses in each block adjoining to each other occupied by natives; 3 blocks with 3 adjoining houses in each block were occupied by natives. After the first cycle of simulation, we find a large shift in the proportion of populations of natives in the city population ranging from 4 blocks with 2 houses adjoining to each other. We find 24 blocks of 3 houses adjoining to each other; 2 blocks with 4 houses adjoining to each other; 7 blocks of5 houses adjoining to each other; 4 blocks of 7 houses adjoining to each other; 2 blocks of 8 houses adjoining to each other, and 1

block of 11 houses adjoining to each other; and 1 block of 12 houses adjoining to each other have been occupied by native group. After the second run of the simulation, we find a huge shift in the population of the natives that varies from 4 blocks of 3 houses. There are 2 blocks of 10 houses, a block of 16 houses, a block of 17 houses, a block of 18 houses, a block of 19 houses, and a block of 21 houses adjacent to each other. After the 3rd cycle, the shift is dramatic where we find several houses consisting of 2 blocks with 11, 12, 16, 18 houses adjoining to each other; 3 blocks of houses are entirely (i.e., 21 houses) populated by the natives and one block consisting of 20 native houses.

FIGURE 3: Landscape After the Third Cycle of 5% Initial Occupancy by the Native Population (Green Cells Show the Native Group)

CONCLUSIONS

Our paper provides a visual illustration of changes in the micro level and their significance at the macro level. Certainly, all of the simulation exercises are rooted in the disciplines of mathematics, computers, and interdisciplinary areas such as socio-economic, socio-technological, and socio-political policies. Interactions among different disciplinary areas are important to provide a realistic understanding

of the problems and their solutions, since simulations are based on number of assumptions and the way the problems are framed (Levy and Wilensky, 2011). However, without going in depth of the methodological issues, we have offered an understanding of simulations and agent-based modeling to students. The goal of this paper has been demystifying the complexity by developing and using simulation. We not only show how simple repetitive behaviors following a set of rules can lead to complex behaviors but also they can take different structures. A clear pattern of segregated areas were formed within a small number of cycles even with a small initial population of the native individuals occupying the houses. The simulator thus showed quick emergence of the large-scale effects of individual decisions.

Since a simulation tends to provide a realistic view of the actual structure and behavior, it is not difficult to understand how simulation can map the complex systems properties (Collard, Mesmoudi, Ghetiu, and Polack, 2013). The results could be summarized that the simulation shows that the macrostructure is more than the sum of the micro-parts, since residents were following simple a behavior, but this behavior led to the segregation of the communities. Secondly, the simulation shows the emergent behavior of the complex system, which could not be known simply examining the micro-level. And finally, the simulation leads to the conclusion that often a reductionist approach falls short in understanding the behavior of the system as a whole.

REFERENCES

Anderson, G. R., Mukherjee, A., and Onder, N. 2009. Traversing and Querying Constraint Driven Temporal Networks to Estimate Construction Contingencies. *Automation in Construction* (18:6), 798-813.

Banks, J., Carson, J.S. II, Nelson, B.L., and Nicol, D.M. 2009. *Discrete-Event System Simulation (5th Edition)*, Prentice Hall, Upper Saddle River, NJ.

Byrne, P.J., Cathal, H, Blake, P., and Liston, P. 2013. A Simulation Based Supply Partner Selection Decision Support Tool for Service Provision in Dell. *Computers and Industrial Engineering* (64:4), 1033-1044.

Chou, J. 2011. Cost Simulation in an Item-Based Project Involving Construction Engineering and Management. *International Journal of Project Management* (29:6), 706-717.

Collard, P., Mesmoudi, S., Ghetiu, T., and Polack, F. 2013. Emergence of Frontiers in Networked Schelling Segregationist Models. *Journal Complex Systems* (22:1), 35-59.

Coveney P., and Fowler, P. 2005. Modelling Biological Complexity: A Physical Scientist's Perspective. *Journal of Royal Society Interface* (2:4), 267-280.

Levy, S.T., and Wilensky, U. 2011. Mining Students Inquiry Actions for Understanding of Complex Systems. *Computers & Education* (56:3), 556-573.

Ottino, J. 2003. Complex Systems. *AIChE Journal* (49:2), 292-299.

Rodger, J.A. 2012. Toward Reducing Failure Risk in An Integrated Vehicle Health Maintenance System: A Fuzzy Multi-Senor Data Fusion Kalman Filter Approach for IVHMS. *Expert Systems with Applications* (39:10), 9821-9836.

Rojas, E., and Mukherjee, A. 2006. A Multi-Agent Framework for General Purpose Situational Stimulation in the Construction Management Domain. *Journal of Computing in Civil Engineering* (20:6), 1-12.

Schelling, T. 1978. *Micromotives and Macrobehavior*. New-York: Norton, New York.

Siemens, 2014 Plant Simulation. http://www.plm.automation.siemens.com/en_us/products/tecnomatix/plant_design/plant_simulation.shtml. [Retrieved: April 5, 2014].

Tang, P., Cass, D., and Mukherjee, A. 2013. Investigating the Effect of Construction Management Strategies on Project Greenhouse Gas Emissions Using Interactive Simulation. *Journal of Cleaner Production* (54), 78-88.

Uzzafer, M. 2013. A Simulation Model for Strategic Management Process of Software Project. *Journal of Systems and Software* (86:1), 21-37.

About the Authors

Sandip C. Patel, Ph.D., a US Fulbright research scholar for 2014-15, is an Associate Professor in the Information Science & Systems Department at the Morgan State University. He has published more than 23 refereed journal papers and 14 conference papers since 2004. Dr. Patel is on the editorial board for six journals and has been a reviewer for 15 journals. Dr. Patel has served as a review panelist for the National Science Foundation's grant of $20 million on cybersecurity. His research has been published in journals such as *Communications of the ACM* and *International Journal of Business Information System*.

Ganesh Bhatt, Ph. D., is a professor in the Department of Information Science & Systems at Morgan State University. His articles have been published in various journals such as *Journal of Management Information Systems, Communications of the ACM, Information & Management, Decision Support Systems, OMEGA, International Journal of Production & Operation Management*, etc.

Ali Emdad, Ph.D., is Professor and Chair of the Department of Information Science and Systems. He received a Ph.D. degree from Case Western Reserve University and has completed the executive management and leadership programs at Harvard and Yale Universities. Dr. Emdad has published more than 50 articles in his research interests of IS strategy; Project Management; education issues- national and global perspectives. Dr. Emdad has received the University's Iva G. Jones Medallion Award, the highest honor given to a faculty for outstanding teaching, scholarship, and service. In 2006, he was awarded a Fulbright Scholarship for teaching and research.