Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

ISSUES IN JAVA-BASED CONTINUOUS TIME STEP PHYSICAL MODELLING

Lisa A. Schaefer
Philip M. Wolfe

Department of Industrial Engineering
Arizona State University
Tempe, AZ 85287, U.S.A.

ABSTRACT

This paper discusses the problems involved in developing a Java based simulation model of autonomous entities that can navigate themselves in 2-dimensional space. We develop some ideas for solving those problems. The ideas mentioned in this paper can be applied to simulations that have fuzzy logic for navigation, encapsulation for object-oriented simulation, many instances of objects, or statistically complex results. We cite reasons for distributing a simulation among several computers and propose several topics for future research.

1 INTRODUCTION

An innovative approach to defining the paths of birds in an animated cartoon flock was invented by Craig Reynolds (Reynolds 1987). The flock is a simulated particle system in which birds are treated as individual objects. Each bird is an independent actor, or intelligent agent, that navigates according to rules on how to avoid nearby obstacles. An example of a set of rules that could be used to navigate a flock of birds is shown below. This rule set was used in an experiment with a herd of mobile robots that mimicked the concept of cartoon birds, except with actual moving entities on a floor in a lab rather than on a computer screen (Mataric 1995).

```
If robot is in path
   If at the right only
      turn left, go forward
   If at the left only
      turn right, go forward
   If on both sides
     Wait
```

There is still much left to be explained with regards to this rule set. How much should the bird turn? How close must the other bird be before we take it into consideration? How fast should the bird fly? There are also several issues regarding the system which are not addressed. What direction are the other birds flying? What if a potential collision is head-on rather than rear-end? How crowded is the system? Will these rules work if a bird is crowded by birds coming head-on?

We experimented with these rules by simulating a flock of autonomous agents and varying parameters in each model according to a central composite experimental design. The factors were: the size of the navigational area, the percentage of that area occupied by agents, the variation of the directions the agents were traveling (as opposed to traveling parallel to each other), and the speed the agents desired to travel if no other agents were impeding their progress. We also experimented with fuzzifying the navigation rules. The simulation program was written in Java, an object-oriented language, in which each bird-like entity was modeled as an instance of an "agent" object. Since this is a time-step simulation, instead of a discrete event simulation, a simulation package would not provide much functionality beyond source code, except possibly output analysis.

There are several types of systems that could be simulated with a software architecture similar to the architecture used for this research, and thus may have issues similar to those addressed in this paper. Research on motor vehicles that drive themselves requires complex algorithms to calculate routes and prevent collisions in real time. Pedestrians and vehicles interacting at an intersection could be modeled as a particle system. A manufacturing application for particle system algorithms is a model of automated guided vehicles on a factory floor.

We used the simulation to obtain results pertaining to the efficiency and effectiveness of the system with regard to two performance measures: effective speed and amount of potential collisions. The values of the performance measures were recorded periodically throughout the simulation, then these results were regressed to determine equations of the performance measures as functions of the experimental design factors.

The following sections describe the issues that arose during the development of this research analysis.

2 ISSUES

Five categories of problems arose while developing the simulation problem for this research. These five issues, along with the solution we implemented, are described in this section.

2.1 Collision Checking

During each simulated time step, agents executed a set of navigation rules to decide in which direction to travel. Each instance of an agent existed in a Java vector. Placing the agents in a vector not only facilitated ensuring every agent executed its rules during each time step.

The rules do not guarantee that the entities will not collide with or overlap each other. Therefore a collision check calculation was required at each time step, separate from the rule execution. Placing agent locations in a vector assisted in validating the simulation at the end of each time step by enabling the programmer to check the locations of each agent to check if any agents overlapped. In preliminary simulation runs, overlapping agents did exist. Several calculation methods were attempted to correct this problem, however only one was somewhat successful.

One idea was to model individual entities as cells which exist over a given amount of area. Cellular automata models of traffic have been developed (Blue et. al. 1996, Nagel and Rasmussen 1995). The term "cellular automata" comes from defining discrete squares of space, or cells, and each cell has a set of rules that governs its state at each time step automatically. Since the cells occupy areas rather than points, the problem of entities colliding due to modeling them as points, rather than rectangles, is eliminated. However, the difficulty with implementing a cellular automata model is when modeling several entities that occupy different amounts of area. Consider the example shown in Figure 1 of a vehicle-pedestrian system in which each pedestrian occupies one cell. Even if vehicles are modeled as many-celled objects, the difficulty of modeling turning vehicles arises since turning vehicles do not occupy rectangular space parallel to roadways in real world systems. Some type of diagonal rectangular space must be defined or approximated.

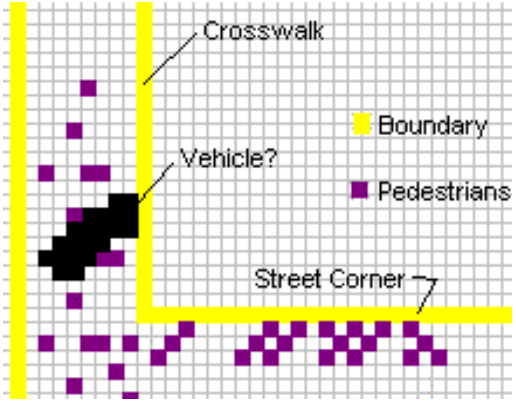
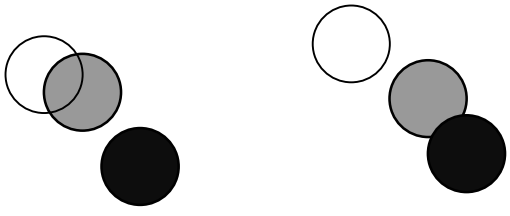


Figure 1: Pedestrians and Vehicles Modeled as Discrete Cells

Thus, the problem of overlapping entities also exists in cellular models with entities of varying sizes. Since the cellular model does not have a means for adequately representing many-celled entities, we decided to use a point-location model within an object-oriented language. The advantage of using an object-oriented approach to model pedestrians is that each object can have an attribute, or variable, which defines its size and shape. Thus pedestrians, vehicles, birds, and any other entity can easily be differentiated from each other.

In our point-location model, we tried moving overlapping agents slightly away from each other so they were not overlapping, but then some were moved on top of other nearby agents as in Figure 2.



a) overlapping agents b) moved agents

Figure 2: Agents Before (a) and After (b) Separating Overlapping Agents

We tried a trigonometric calculation (see Figures 3 and 4), in which each agent checked its path to its next location with the paths of the other agents, but it was difficult to determine how to check agents that had already been moved due to effects of an agent it had been compared with previously.

In Figures 3 and 4, the closest potential collision for the black agent is the white agent. However, the closest potential collision for the white agent is the grey agent. If the black agent is the first agent to traverse the vector of agents to locate its collision mate, the black and white agents will be moved next to each other as in Figure 3, however the white and grey agents should be moved next to each other as in Figure 4 and the black agent does not collide with any agent.

We finally settled with the concept of mini-steps. One step is the distance an agent travel while it calculates one iteration of its maneuver-checking rules. With this method, each agent is moved a fraction of its step length iteratively during each time step until it either reaches another agent or its step length. See Figure 5. Although this stopped any agent involved in a collision in a realistic location, this increased run time on the order of n^2, n being the number of mini-steps that make up a whole step.

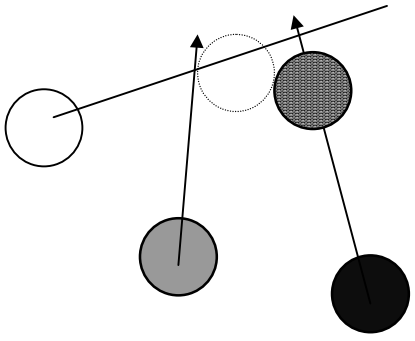


Figure 3: Black Agent Moved as if it Collided with the White Agent

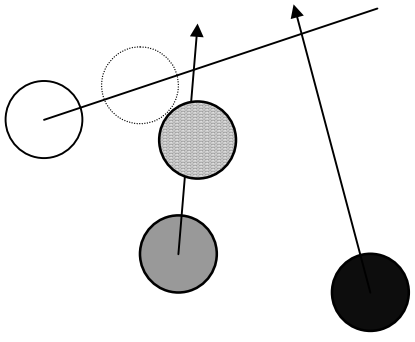


Figure 4: White and Grey Agents Collide First

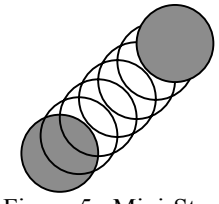


Figure 5. Mini-Steps

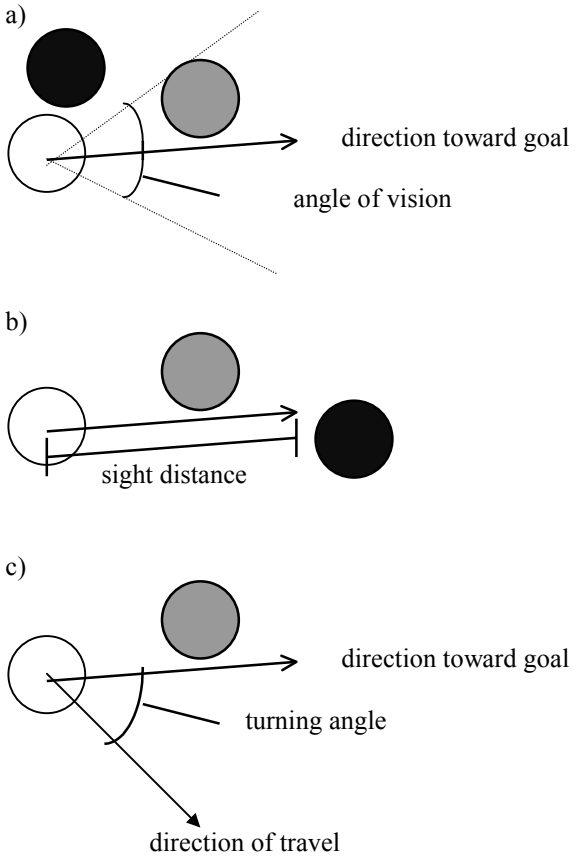


Figure 6. a) Angle of Vision, b) Sight Distance, and c) Turning Angle

2.2 Fuzzification

Originally there were seven factors in our experimental design, however we decided that three of them had narrow logical ranges and that implementing values outside these ranges would be illogical. These three factors were: angle of vision (the angle within which other agents can be detected), sight distance (distance within which other agents can be detected), and turning angle (angle the agent turns away from goal location if another agent is detected). An agent's goal location its destination, which may represent a machine to which the agent is delivering an unfinished product for processing. See Figure 6 for further clarification of the meaning of these factors.

Instead of varying angle of vision, sight distance, and turning angle within an experimental design, we considered using fuzzy logic to vary these values within the agents' navigation rules. Fuzzy logic involves the use of sets or ranges of values to quantify a value instead using one crisp number.

When programming a flock of mobile robots, it may be difficult to determine whether fuzzy logic is the best method to use in the navigation algorithms. Fuzzy logic may be inefficient, requiring complicated calculations to determine the crisp outcome, such as the speed to travel or angle to turn. Fuzzy logic also may not give better or more accurate results, e.g. using an angle of 45^{o} all the time may be just as good as a function that requires a series of calculations that usually results in a value of 40^{o} or 50^{o} anyway.

When using fuzzy logic, it is also difficult to determine how membership functions should be developed.

Membership functions define the degree to which a range of values belong to a category. See Tsoukalas and Uhrig (1997) for more information about fuzzy logic and membership functions. It seems that engineering judgment is the best way to determine appropriate membership functions for this particular problem. In one of our sets of navigation rules, each agent used a membership function to determine which agent was the critical agent to avoid based on distance to the other agent and how close the other agent was to being directly ahead, as opposed to being on the side. If the other agent was very close and directly in front of the current agent, it had a high membership value and had a high chance of being the critical agent to avoid. If the other agent was not very close and directly to the right (or left) of the current agent, it had a low membership value and may not have played a critical role in the current agent's collision avoidance maneuvering rules.

Not only is it important to determine what the functions should be, but it may be more important to determine what aspects of the model should be fuzzified and what aspects should not. In this study, possible values to fuzzify were: field of view, turn angle, and view distance. Field of view was determined to be inappropriate for fuzzification because the range of feasible values is small, thus varying the values would not give much insight into the problem.

2.3 Encapsulation

The simulation architecture for this experiment included many instances of an Agent object and one instance of a Field object which represented the two-dimensional space over which the agents navigated. The instances of the Agent object contained parameters particular to each agent and the methods required for navigation. The Field object contained system parameters and simulation methods.

During simulated navigation, each agent executed a set of rules that were dependent upon the locations of nearby agents. Each agent's location was a private x-y attribute. After each simulated time step, agents "broadcast" their location by updating their location within a vector of all agent locations. The location vector exists within the Frame object.

Changing the parameters for the experimental design required extensive message passing. Parameters in each instance of the Agent object had to be changed. It is possible to place these attributes in the Field object, which is accessible to all agents, however then each agent would have to access that parameter every time it executed its rules. It is uncertain which method is better, especially if the simulation were distributed on several virtual machines with the field on a different machine than the agents. If this were the case, perhaps a serialized object could contain a data packet that was passed to the agent whenever it requested information from the field.

2.4 Distributing Computation to Improve Run Time

The mini-steps caused the run time to increase by the order of n^2. Thus the simulations with many instances of agent objects took days to run. This was partly solved by distributing the simulation among several computers. The source code was written so some of the experimental design treatments were in one set of code, other treatments were in another set of code and run on a different computer. Thus different runs were performed simultaneously on different computers, which worked well for our experiment. In our particular experiment, the results from each treatment were not dependent on each other, thus it was not necessary to have the computers networked for this particular experiment. Instead, the simulation was able to run independently on each computer with different input sets for each treatment.

Distributing the computational load of a simulation among several computers may be a quick fix to get the job done, however if the code is to be reused, it may be worthwhile to determine how to make the simulation more efficient. One method we did not attempt was to place each instance of an agent on its own virtual machine on several computers across a network. We believed that it may be an interesting experiment in itself to determine the maximum number of agents per processor possible while still improving run time. This type of simulation would require different communication protocols than a simulation on one processor and was beyond the scope of our analysis.

The authors considered creating each agent as its own thread. Unfortunately the result was that some agents executed their rule sets many times in a row, while others did nothing. The relative navigation speed of the agents cannot be controlled if each agent is a thread.

2.5 Obtaining Reasonable Results

We hypothesized that the regression model would be quadratic in form with respect to speed and diversity of agents' travel direction, and the performance measures would be inversely proportional to floorspace utilization.

We used regression analysis with MiniTab to obtain the equation to describe the flow theory for the autonomous entities. Since we knew that the reciprocal of at least one of the factors was important in the analysis, we included the reciprocals of all factors in our analysis to determine if they would give the equation a better fit to the simulation output. This resulted in obtaining a very high R^2 value, around 0.9.

However when we plotted our equation, we discovered that the predicted values were orders of magnitude out of the range of all simulation data points. Upon further examination of the results, we realized not all the terms in

the equation made intuitive sense. Therefore we removed some of the interaction terms from our analysis and developed more reasonable equations. The R^2 was only around 0.6, not as high as we wished. However we learned the lesson that a high R^2 does not always mean the results are adequate. One of the resulting equations is plotted in Figure 7.

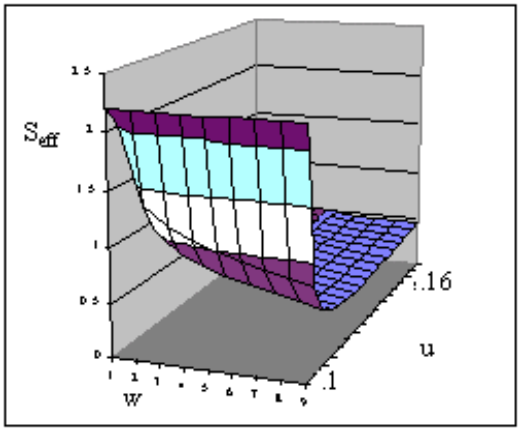


Figure 7. Appropriate Plot for Describing Effective Speed as a Function of Floorspace Utilization (u) and Diversity of Agents' Travel Direction (w)

This plot tells us that in a system in which the agents travel in more widely varying directions, the average effective agent speed is lower than in systems in which agents travel in directions more parallel to each other. As the floorspace becomes more crowded, the average effective agent speed is again reduced. Intuitively the trends shown in the plot are reasonable and coincide with the hypothesized form of the regression model. The details of the results of this model are beyond the scope of this paper and can be found in Schaefer et al. (2000).

3 CONCLUSION

When creating a model of a system with many moving objects, one must be careful when choosing the manner in which to represent entities. If the entities are modeled as points but they represent areas in the real system, the rules must account for appropriate collision detection, otherwise the simulation results may not be valid. If the entities are modeled as cells, the rules may need to account for entities of varying sizes.

It may be important to determine what aspects of a model should be fuzzified and what aspects should not. One must consider what the logical ranges for fuzzy values might be. Sometimes the ranges may be very narrow, thus a fuzzy value may not be more accurate than a crisp value. Fuzzy logic may require complicated calculations to determine the crisp outcome.

Several other lessons were learned. It may be difficult to decide in which object to place attributes if several types of objects must access that attribute. One must use good judgment when deciding which interaction terms to include in a regression model. Distributing the computational load of a simulation among several computers may be a quick fix to get the job done, however if the code is to be reused, it may be worthwhile to determine how to make the simulation more efficient.

The authors intend to continue work in this area by implementing the rule sets on several virtual machines to simulate a distributed system of vehicle traffic approaching an intersection, then extending the research to a set of several intersections. We expect many more issues to come up as we develop a distributed simulation of a system similar to the one used in this research, however that will be a topic for a future Winter Simulation Conference paper.

REFERENCES

Blue, V., F. Bonetto, and M. Embrechts. 1996. A Cellular Automata Model of Vehicular Self-Organization and Nonlinear Speed Transitions. *Proceedings of the Transportation Research Board 75th Annual Meeting*. Paper No. 961336.

Mataric M. 1995. Designing and understanding adaptive group behavior. *Adaptive Behavior,* 4 (1): 51-80.

Nagel K. and S. Rasmussen. 1995. Traffic at the edge of chaos. *Transportation Science*, 28 (2): 222-235.

Reynolds, C.W. 1987. Flocks, Herds, Schools: A Distributed Behavioral Model. *ACM SIGGRAPH Computer Graphics*. 21 (4): 25-34.

Schaefer L. A. and C. W. Kirkwood. 2000. Model Selection for Engineering Design: A Multiobjective Decision Analysis Approach, with an Application to Simulating Mobile Physical Agents. Submitted to *IEEE Transactions on Systems, Man and Cybernetics.*

Schaefer L. A., D. C. Montgomery and P. M. Wolfe. 2000. Flow Theory for Flocks of Autonomous Physical Agents. Submitted to *Transportation Science.*

Tsoukalas L. H. and R. E. Uhrig. 1997. *Fuzzy and Neural Approaches in Engineering*. John Wiley and Sons, Inc. New York.

ACKNOWLEDGMENTS

This work was partially supported by the Federal Highway Administration Office of Advanced Research.

AUTHOR BIOGRAPHIES

LISA A. SCHAEFER is a Ph.D. candidate in industrial engineering at Arizona State University. She received her B.S.E. and M.S. in civil/ transportation engineering from

Arizona State University. She is a member of SCS and IIE. Her interests include O-O simulation, transportation, and distributed simulation. Her email and web addresses are <LschaeferAZ@hotmail.com> and <www.public.asu.edu/~schaefer>.

PHILIP M. WOLFE is a professor and former Chair of the Department of Industrial Engineering at Arizona State University. He received his B.S. in industrial engineering and B.S. in business administration from the University of Missouri. He also has an M.S. and Ph.D. in industrial engineering from Arizona State University. Previous positions were at the Garrett in Phoenix, Arizona, Oklahoma State University, and Motorola in Phoenix, Arizona. His interests are related to computer applications and manufacturing. His email and web addresses are <wolfe@asu.edu> and <pwolfe.eas.asu.edu>.