NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

**AGENT-BASED SIMULATION
OF A MARINE INFANTRY SQUAD
IN AN URBAN ENVIRONMENT**

by

Arthur R. Aragon

September 2001

Thesis Advisor: Neil Rowe
Second Reader: Chris Eagle

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) Agent-Based Simulation of a Marine Infantry Squad in an Urban Environment			**5. FUNDING NUMBERS**
6. AUTHOR(S) Aragon, Arthur R.			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

This thesis research focused on the design, development, and implementation of an agent-based simulation of a Marine infantry squad in an urban environment. The goal was to design an autonomous-agent framework that could model a combatant's decision cycle. A squad entity comprised of these agents was created to explore the idea of team dynamics and the balance between meeting individual goals and team goals. The agents were placed in a two-dimensional, discrete-state, simulation world with a simple model of urban infrastructure. The squad goal was to patrol through the environment using checkpoints. The individual agent goals were to move to a destination and maintain the squad formation. The critical issues of agent movement were collision detection/avoidance, goal managing and forward planning. Distinguishing the agents by their role in the squad allowed a single agent to act as the squad leader. This agent was given the ability to plan a path to accomplish the squad's overall goal as a series of sub-goals, which was successful in getting the majority of the agents to their checkpoints in squad formation. The design of the simulation program facilitates further research in using autonomous agents to model small-units in an urban environment.

14. SUBJECT TERMS Autonomous agents, Computer Simulation, Urban Combat, Military Operations in Urban Terrain, Java, Software Engineering, Model-View-Controller Architecture			**15. NUMBER OF PAGES** 161
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

AGENT-BASED SIMULATION OF A MARINE INFANTRY SQUAD IN AN URBAN ENVIRONMENT

Arthur R. Aragon
Captain, United States Marine Corps
B.S.M.E., United States Naval Academy, 1996

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

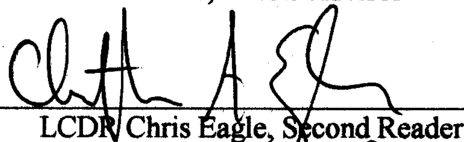
NAVAL POSTGRADUATE SCHOOL
September 2001

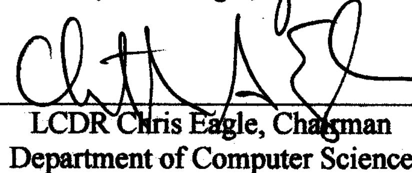
Author:

Arthur R. Aragon

Approved by:

Neil Rowe, Thesis Advisor


LCDR Chris Eagle, Second Reader


LCDR Chris Eagle, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis research focused on the design, development and implementation of an agent based simulation of a Marine infantry squad in an urban environment. The goal was to design an autonomous-agent framework that could model a combatant's decision cycle. A squad entity comprised of these agents was created to explore the idea of team dynamics and the balance between meeting individual goals and team goals. The agents were placed in a two-dimensional, discrete-state, simulation world with a simple model of urban infrastructure. The squad goal was to patrol through the environment using checkpoints. The individual agent goals were to move to a destination and maintain the squad formation. The critical issues of agent movement were collision detection/avoidance, goal managing and forward planning. Distinguishing the agents by their role in the squad allowed a single agent to act as the squad leader. This agent was given the ability to plan a path to accomplish the squad's overall goal as a series of sub-goals, which was successful in getting the majority of the agents to their checkpoints in squad formation. The design of the simulation program facilitates further research in using autonomous agents to model small-units in an urban environment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION....1
A. THESIS STATEMENT....1
B. PURPOSE....1
C. THESIS GOALS....1
D. THESIS ORGANIZATION....2
II. BACKGROUND....3
A. GLOBAL URBANIZATION AND MILITARY OPERATIONS....3
B. THE CHALLENGES OF URBAN WARFARE....4
C. TRAINING FOR URBAN WARFARE....5
D. MODELING, SIMULATION AND AUTONOMOUS AGENTS....6
E. ISAAC....8
F. ARCHIMEDES....9
III. SIMULATION DEVELOPMENT....11
A. STATEMENT OF REQUIREMENTS....11
B. REQUIREMENTS SPECIFICATIONS....11
C. SOFTWARE ARCHITECTURE....12
1. Models....13
2. View....14
3. Controller....16
D. SIMULATION ENVIRONMENT....16
IV. AGENT DEVELOPMENT....19
A. AGENT FRAMEWORK....19
B. STATE VARIABLES....20
C. GOALS AND GOAL MANAGING....21
D. AGENT ACTION....22
1. Basic Movement....22
2 Collision Detection/Avoidance....23
3. Path Planning....24
E. SQUAD RELATIONSHIP....28
F. SQUAD VS. INDIVIDUAL GOALS....29
V. SIMULATION ANALYSIS....31
A RUNNING THE SIMULATION....31
B. SIMULATION RESULTS....31
1. The Scenario....31
2. Squad and Individual Agent Interaction....31
VI. FUTURE WORK AND CONCLUSIONS....35
A. FUTURE WORK....35
1. Develop Agent Personalities....35
2. Develop Agent Actions....35

3. Develop different agent types 35
4. Develop the infrastructure 35
5. Develop the User Interface 35
6. Create Smarter Agents 35
B. CONCLUSIONS 36

APPENDIX A. JAVA SOURCE CODE 37

LIST OF REFERENCES 141

BIBLIOGRAPHY 143

INITIAL DISTRIBUTION LIST 145

LIST OF FIGURES

Figure 1. Model-View-Controller Architecture. 12
Figure 2. Class Diagram of Model Objects. 13
Figure 3. Infrastructure Class Hierarchy. 14
Figure 4. The View Component's Display...15
Figure 5. Agent Framework. 19
Figure 6. Agent Movement Constraints. 23
Figure 7. Path Search Node Network. 25
Figure 8. Best-First Search (Shortest Path). 26
Figure 9. Best-First Search (Cover and Concealment Benefit). 27
Figure 10. Squad-Marine Agents' Class Relationship. 28
Figure 11. Squad Moving to Third Checkpoint. 33

THIS PAGE INTENTIONALLY LEFT BLANK

DISCLAIMER

The computer program developed in this research was exploratory in nature. It cannot be considered verified or validated. Any application of this program outside of experimental research is not recommended and is done at the risk of the user.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

U.S. military operations in urban terrain have been directly affected by global urbanization. The level of training and research in urban tactics must correspond to its growing relevance on the global scene. Simulation can provide great benefits to the study of urban tactics at the small-unit level.

This thesis research focused on the design, development and implementation of an agent based simulation of a Marine infantry squad in an urban environment. The primary goal was to design an autonomous-agent framework that could model a combatant's decision cycle and create a squad of these agents working to accomplish both individual and team goals. The agents were placed in a two-dimensional, discrete-state, simulation world with a simple model of urban infrastructure. The squad goal was to patrol through the environment using checkpoints. The individual agent goals were to move to a destination and maintain the squad formation.

The critical issues of agent movement were collision detection, collision avoidance, goal managing, and forward planning. Distinguishing the agents by their role in the squad allowed a single agent to act as the squad leader. This agent was given the ability to plan a path to accomplish the squad's overall goal as a series of sub-goals, which was successful in getting the majority of the agents to their checkpoints in squad formation.

The design of the simulation program facilitates further research in using autonomous-agents to model small-units in an urban environment to analyze the efficiency of current tactics and to experiment with new ones.

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENT

I would like to thank my advisor, Dr. Neil Rowe, for his guidance and patience during this research.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. THESIS STATEMENT

A constructive agent-based simulation of a Marine infantry squad in an urban environment can serve as a useful tool for the warfighter. It can be used to study the dynamics between the individual Marine and the squad to help define the boundaries between self-preservation and teamwork. It can be used to analyze the efficiency of current tactics, techniques, and procedures used in urban combat. It can provide the repeatability and statistical data that live simulations lack. Finally, it can be used to experiment with new and innovative tactics against the potential threats of the future.

B. PURPOSE

The purpose of this thesis is to take the first step in creating such an agent-based simulation. The primary effort will be to create an abstract model of a Marine infantry squad in an urban environment. Engineering autonomous agents in a constructive, discrete-state simulation program using the Java object-oriented programming language is how this will be accomplished. The simulation will focus on the motions of the squad and its individual Marine agents in order to study the team dynamics and performance of the squad in accomplishing the simple task of patrolling through an urban area.

C. THESIS GOALS

The overall goals of this thesis are:

- To engineer an autonomous Marine agent with individual goals and an individual personality;
- To model a Marine infantry squad comprised of these autonomous Marine agents;
- To engineer an object-oriented discrete-state simulation with a 2D display to depict the actions of the agents in the simulation;

- To study the balance between individual agent goals and squad goals;
- To model simple squad tactics; and
- To identify emergent behavior of the agents in the simulation.

D. THESIS ORGANIZATION

This thesis is organized into the following chapters:

- Chapter II: Background. Identifies the root of the problem being addressed and how a simulation tool that models urban operations at the squad level can assist in evaluating current tactics and experimenting with new ones. Explains the proposed approach in developing the simulation and the benefits behind that approach
- Chapter III: Simulation Development. Describes the software engineering process from requirements to software architecture.
- Chapter IV: Agent Development. Describes the development of the autonomous agents from a high-level framework to the software mechanisms that allow the agent to assess their environment, make decisions and act on those decisions.
- Chapter V: Simulation Analysis. Describes the current simulation status and analysis of squad and individual agent behavior.
- Chapter VI: Future Work and Conclusions.

II. BACKGROUND

A. GLOBAL URBANIZATION AND MILITARY OPERATIONS

One of the most significant global trends characterizing the last century is the urbanization of the world's population. According to recent UN reports, an estimated 150,000 people are added to the urban population of developing countries every day. Within the next decade, more than 50 percent of the world's population will live in urban areas (UN, 1994); this figure is expected to increase to over 60 percent by the year 2025.

Some areas of the world have seen phenomenal population increases in very short periods of time. Japan, Mongolia, and the continent of Africa have all seen their urban population increase by more than 25% in less than a 50-year period. (United Nations Population Division 1997). China has gone from 192 million people living in urban environments to 377 million in a 16-year period from 1980 to 1996. The most dramatic increase in urban population occurred in the Democratic People's Republic of Korea, which saw an increase from 17.7 percent to 61.2 percent from the years 1953 to 1995(United Nations Population Division 1997).

In more cases than not, rapid urbanization greatly exceeds the growth rate of urban infrastructures. This will lead to an eventual lack of energy resources, housing, and job opportunities for those immigrating to urban centers. The end result is an increase in poverty and crime, which sets the stage for regional instability and potential crises.

U.S. military operations in urban terrain have been directly affected by urban population growth over the last 50 years. From the landing at Inchon, the battle for Seoul, and Hue city, to operations in Beirut, Panama, Haiti, Somalia, and Bosnia, the U.S. military has seen a shift from operations in open country to operations in urban centers ranging from small villages and towns to major metropolitan cities.

B. THE CHALLENGES OF URBAN WARFARE

Up through World War II there were generally two ways to deal with urban areas. The first was to outright avoid them. Bypassing a city and controlling its external nodes was an inexpensive way of controlling the city itself. Controlling these external nodes allowed a military force to cut supply and communication lines to that city without having to enter it. The second method was to reduce the city by aerial or artillery bombardment and then enter and destroy it with little or no regard for collateral damage or civilian casualties.

There were genuinely good reasons why military leaders chose to deal with urban areas in these two ways. Urban warfare, since the Middle Ages, has always been more expensive and complex than warfare in open country. It is very resource and man-intensive, casualty rates are high, and operations are usually painstaking and very time-consuming. Today's urban environment adds many more unique challenges.

First, urban environments provide a three-dimensional ground threat not seen in conventional non-urban settings. Where control of the ground in non-urban terrain is defined by surface area, control of the ground in an urban environment must be defined in terms of volume. Multi-level buildings, basements, sewer systems, subway systems, all provide a third dimension. For a military force to truly "control" an area in an urban environment, it must control this third dimension.

Secondly, a technological advantage in weaponry is neutralized in an urban environment. The military has used technology to gain the advantages of increased range and lethality in weapon systems. These advantages are not only neutralized in urban environments but may serve as disadvantages. One characteristic common to platforms with increased ranges is a minimum target distance. In the close-quarter conditions of urban warfare, a military force is not likely to have the minimum distance between itself and the enemy to take advantage of these long-range weapons. Additionally, the advantage of increased lethality of certain weapons equates to an increase in the effective casualty radius of its munitions. With the close-quarter conditions of the urban environment, using these weapons can lead to fratricide or undesired collateral damage.

Third, the close-quarter conditions themselves serve as a challenge. Close-quarter conditions reduce a military unit's ability to disperse itself, making it more vulnerable to high casualty rates when area-target weapons like grenades, mortar rounds, or other explosives are used against it. Close-quarter conditions increase the risk of fratricide, as friendly units will operate much closer together and in more chaotic conditions. Close-quarter conditions increase the challenge of locating and targeting the enemy, and reduce the frontal coverage of a single unit, creating the need for more forces in the area of operation.

Finally, the urban environment has obstacles not encountered in non-urban settings. Buildings, rubble, cars, fences, windows, doors, walls, and furniture are just a few. Environmental factors to be noted are electrical hazards from power lines and generators, natural gas hazards from gas lines and heating systems, and gasoline hazards from gas stations and automobiles. In addition to the obstacles themselves, the material of which they are made can affect the tactics a military unit practices. For example, some building materials may preclude the use of specific weapons due to fire hazard or the risk of fratricide.

These are a few of the many challenges a military force is faced with in urban combat. Armed conflict is no longer defined solely by full-scale wars, but by a spectrum of operations ranging from peacekeeping and other low-intensity conflict to high-intensity combat operations. In recent military operations urban centers could not be avoided. In addition, global opinion has become less tolerant of excessive collateral damage and non-combatant casualties. Leveling an urban center to rubble has become unacceptable as a military course of action. New methods must be developed.

C. TRAINING FOR URBAN WARFARE

The level of training in Military Operations in Urban Terrain (MOUT) must correspond to its growing relevance on the global scene. The two greatest challenges in training in MOUT are realism and constancy.

In order for any combat training to be effective it must be realistic. Military units must see and experience the unique challenges associated with the urban environment in realistic combat settings. In order for troops to learn which tactics are effective, they must train in realistic scenarios where casualties are assessed. Obviously, live-fire exercises are not an option. However, the reasonable alternatives like MILES gear (laser-tag) or paint-ball guns are either poor in achieving realism or too expensive to conduct on a regular basis.

MOUT training must also be continuous. The tactics, techniques and procedures for urban warfare are more complex than those associated with operations in non-urban terrain. Therefore, training in an urban environment must be done on a more regular basis and should be the focus of Marine combat training. The problem in today's military is that the training sites for this specific kind of training are few in relation to the number of units needing them; frequency of training is limited to 4 to 5 training exercises a year. MOUT training demands much more attention.

D. MODELING, SIMULATION AND AUTONOMOUS AGENTS

Although there is no substitution for realistic training, computer modeling and simulation (M&S) may greatly help urban warfare training problems. Modeling can be defined as representing reality in a simplified manner such that some lesser details are avoided, but the integrity of dependencies and relationships are maintained. A model is characterized by three attributes. The first is the *Reference* or what the model refers to or represents. The second attribute is the *Purpose* or the intended cognitive reason for the model with respect to its referent. The last attribute is the *Cost-effectiveness* or the benefit of using the model instead of the referent itself. (Rothenberg, 1989)

Computer simulation is defined as the translation of a model from a mathematical or logical form into a working computer program. There are two approaches to simulation: analytical and discrete-state. Analytical simulation uses mathematical analysis to describe the referent. Its limitation is that not everything can be described in terms of mathematical equations. The discrete-state approach attempts to define the

referent in terms of unique states connected to another by events or other low level interactions, which cause transitions from one unique state to another, (Rothenberg, 1989).

Simulation can provide great benefits to the study of MOUT. It can provide vantage points not feasible in live training or real combat because of the limitation of real-life perspective. It can graphically depict the relationships between the agents in the squad and their interaction with the simulation environment. Other benefits include repeatability of scenarios and control over time.

Simulating MOUT at the small-unit level with a focus on close-quarter battle techniques can also serve as a useful tool for small unit leaders. It will allow leaders to determine the effectiveness of existing tactics, techniques and procedures, to learn about the special challenges associated with operating in the urban environment, and to study the team dynamics of a small unit in such an environment. Simulation could also prove to be an inexpensive solution to experimenting with new and innovative tactics.

For this thesis, a Marine infantry squad in an urban combat scenario will be modeled using autonomous agents in a discrete-state simulation. Autonomous agents are encapsulated software objects that take input from their environment, process that input, and then output specific actions that in turn affect and change the environment, (Weiss, 1999). A software agent is capable of autonomous processing and autonomous action in order to meet its design objectives. Using autonomous agents to represent each Marine will facilitate the modeling of difficult cognitive and reactive responses not captured by other modeling techniques, they will make decisions and act independent of the user. This will provide a more realistic perspective in studying the performance of the squad and each Marine in a decentralized setting. Additionally, it will help depict the cause-and-effect relationships between the individual agents and their environment and how the relationships effect the operation of the squad. The real-life details of an infantry squad in a combat scenario will not and cannot be modeled. However, all the important relationships and dependencies between all the components of the overall simulation model will be represented.

Using C++ or Java, the model and its relationships will be implemented as an object-oriented simulation program. It will include a graphical user interface which will allow the user to view the model and the interaction of the components within the model in a graphical representation instead of raw, incomprehensible data. Some of the benefits of using an object-oriented approach to simulation are the comprehensibility of the design, encapsulation of information, extensibility of the program for future development, and modularization of the design, all of which will help make the program easier to test and program errors easier to identify.

E. ISAAC

Irreducible Semi-Autonomous Adaptive Combat (ISAAC) is an agent-based simulation developed by Andrew Ilachinski for the U.S. Marine Corps' Combat Development Command (MCCDC), (Ilachinski, 1997). The focus of Ilachinski's work was to approach ground combat as a Complex Adaptive System of individual autonomous agents. The ISAAC agents, ISAACA's, represent low-level combatants with the ability to adapt to the environment by responding to local information and, based on adjustable "personality" vectors, making decisions advancing them towards a single overall goal to capture the opponent's flag.

Conventional combat models are primarily based on the Lanchester Equations created by F.W. Lanchester in 1914. These equations provided a simple way to adjudicate attrition style combat based only on attrition rates and the sizes of the forces involved but cannot capture the dynamic, non-linear characteristics of real combat or the adaptive behavior of combatant forces. ISAAC's approach provides a higher level of resolution to combat modeling.

The intent behind this thesis is to build on Ilachinski's approach to modeling ground combat with autonomous agents modeling combatants; however, the focus will be on simultaneously modeling the unit to which these combatants belong to explore the relationship between the individual agent and the team. The individual agents will also

have the ability to manage and address individual goals while attempting to accomplish the team's overall goal.

Another aspect to the ISAAC simulation that this thesis will focus on is the idea of agent personalities. Each agent has a "personality" vector of weights assigned to specific environmental information: alive friendly agents, alive enemy agents, injured friendly, injured enemy, own flag, and enemy flag. A numerical weight is assigned to each corresponding information type; the larger the weight, the higher the propensity of the agent is to move towards that type of object. For example, an aggressive agent would have high weights for alive enemy agents, injured enemy, and enemy flag, and would be more interested in moving towards enemy agents than towards friendly agents. For our application, personality attributes should be based more on real-life skills and attributes such as physical fitness, marksmanship, leadership, etc.

F. ARCHIMEDES

In the tradition of ISAAC, ARCHIMEDES is the Marine Corps' current project in combat modeling and simulation. Its primary purpose is to provide a fast-running, flexible simulation architecture that provided variable resolution, variable scenario input, and behavior driven autonomous agents. The ARCHIMEDES platform was more a modeling environment than a model, with an emphasis on allowing the user to tailor the environment and the rules governing agent behavior for a wide range of scenarios. However, the focus of ARCHIMEDES is the development of a modeling tool while the focus of this is the development of the agent model and the relationship between the individual agent and the unit to which it belongs.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMULATION DEVELOPMENT

A. STATEMENT OF REQUIREMENTS

The product of this thesis is a Java application. A modified software engineering approach has been taken to compensate for the time limitations.

The goal of this thesis is to model a Marine infantry squad with individual autonomous agents. Each agent will have individual attributes and goals. The squad itself will be represented as an agent with attributes and goals of its own. This will allow the team dynamics and dependencies to be qualitatively assessed and the effectiveness of tactics, techniques and procedures to be studied. The simulation, known henceforth as the **M**arine **S**quad **C**ombat **S**imulation or MSCS, will be engineered according to the object-oriented programming paradigm in the Java programming language.

B. REQUIREMENTS SPECIFICATIONS

The basic requirements for the Marine Squad Combat Simulation are:

- MSCS will be object-oriented.
- MSCS will utilize autonomous agents to model actors in the simulation
    - Marine agents will be autonomous
    - Enemy agents and noncombatant agents will be autonomous.
- Marine agents will have at least the following attributes:
    - X position;
    - Y position;
    - Heading;
    - Role;
    - Rate of movement;
    - Active goal; and
    - Goal manager.

- All Marine agents will be organized into one squad
- Agents will have goals and decision-making mechanisms
- MSCS will be implemented in the Java programming language
- MSCS will provide a two dimensional interface
- MSCS will be a closed-form, constructive simulation
- MSCS will be engineered for extensibility to allow for the future enhancements

C. SOFTWARE ARCHITECTURE

The architecture of the Marine Squad Combat Simulation is described using only the Conceptual View of the four-view paradigm presented in (Hofmeister, Nord and Soni, 2000). The requirements of the MSCS can be met in a single executable application built using a component-based architecture like the Model-View-Controller. This architecture was chosen because it facilitates the extensibility needed for future enhancements of the program and provides modularity that separates the models from both the interface and the simulation controller, (Burbeck, 1987). It is easy to add, remove, or alter the models without affecting the simulation environment.

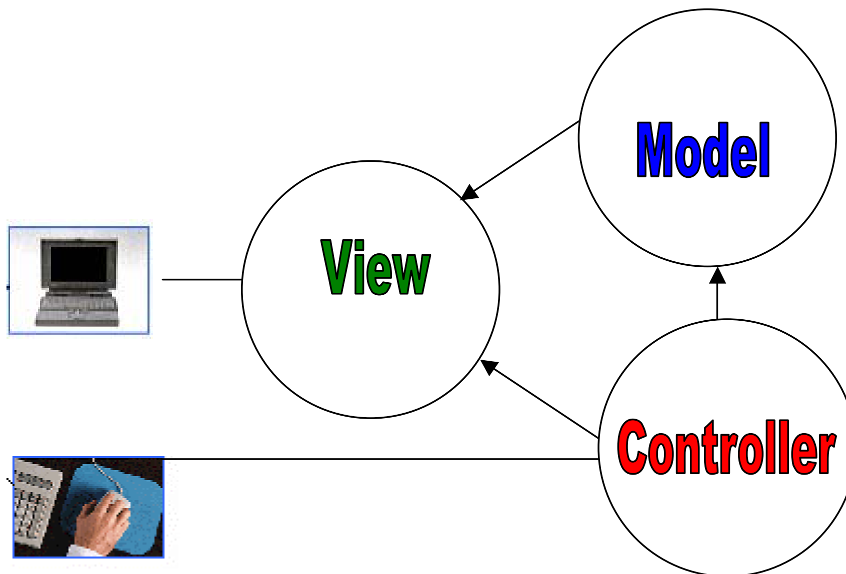


Figure 1. Model-View-Controller Architecture.

1. Models

The model component of the simulation architecture is actually a component of components. It represents all the models in the simulation world. These models will have no association with the View component other than providing information to facilitate the graphical representation of the model.

Every model in the simulation is derived from a single abstract class, *SimObject* (short for simulation object). Each subclass of *SimObject* contains at least an *X* and *Y* position value and a *type* attribute, which classifies the type of model created. Currently the models that can be created are *SimpleAgents* and *BuildingComponents*. From these, more specific objects like *Marine* agents, walls and doors can be created with attributes and methods tailored to those specific models. The Unified Modeling Language, UML, class diagram in Figure 2 shows the hierarchy of objects that represent the models in the simulation, (Eriksson and Penker, 1998).

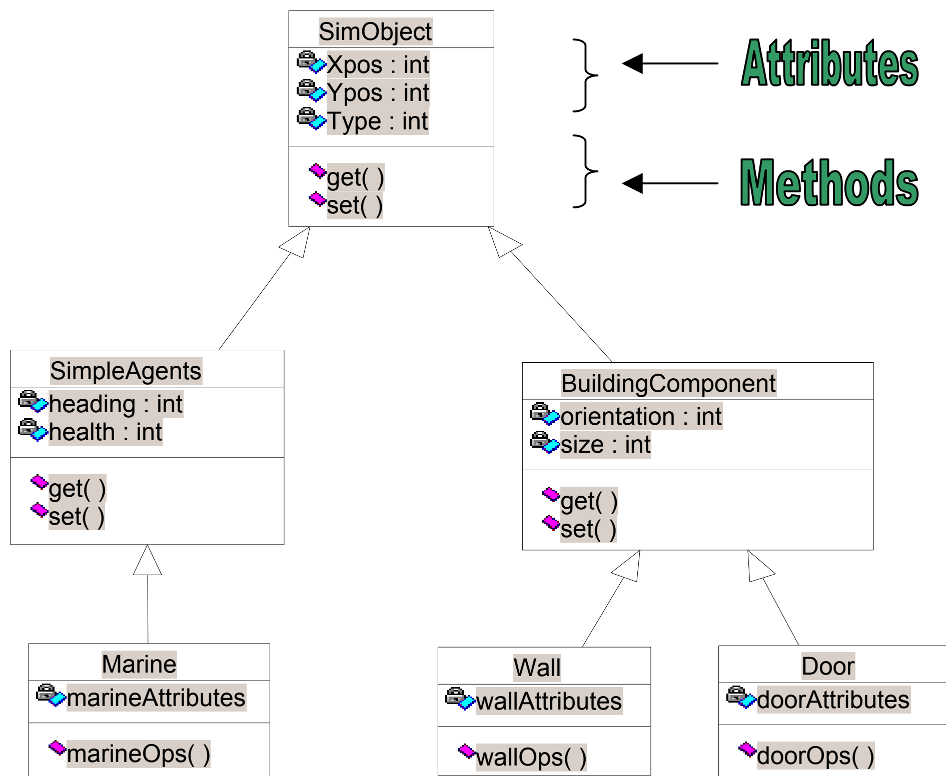


Figure 2. Class Diagram of Model Objects.

These basic models were then encapsulated into larger collections of the models. For example, an abstract base class, *Building*, was created which contains Wall and Door objects. Derivations of the *Building* class, *Building1* and *Building2*, combine different numbers of walls and doors to create different building structures. Furthermore, a class named Infrastructure contains a collection of *Building* derivatives; Figure 3 shows its class relationships.

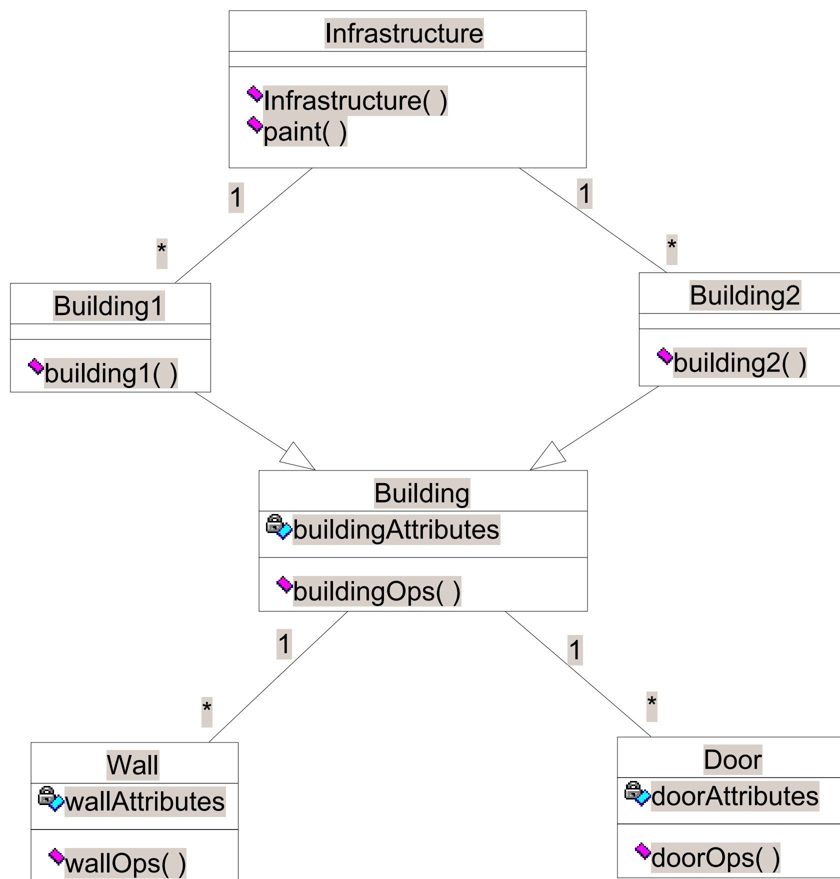


Figure 3. Infrastructure Class Hierarchy.

2. View

The view component is the direct interface between the user and the controller. It is responsible for displaying graphics and data provided by the models and the controller. For MSCS the view component is encapsulated in the *paint()* function of the Java Applet

controlling the simulation. The view component communicates with each of the models through the controller. Each model contains its own *paint()* function, which communicates to the view component the information needed to graphically display each model on the two-dimensional display. Figure 4 shows the display provided by the view component, which graphically depicts the simulation world and models.

The environment for the simulation program is modeled as a two dimensional system of grid squares. The grid squares have dimensions of 5 x 5 pixels. Agents are represented by cyan (light blue) squares. Walls are represented as several gray squares. Doors are represented by gray-outlined squares. Only one *SimObject* can occupy each grid square at a time. Orange squares represent checkpoints.

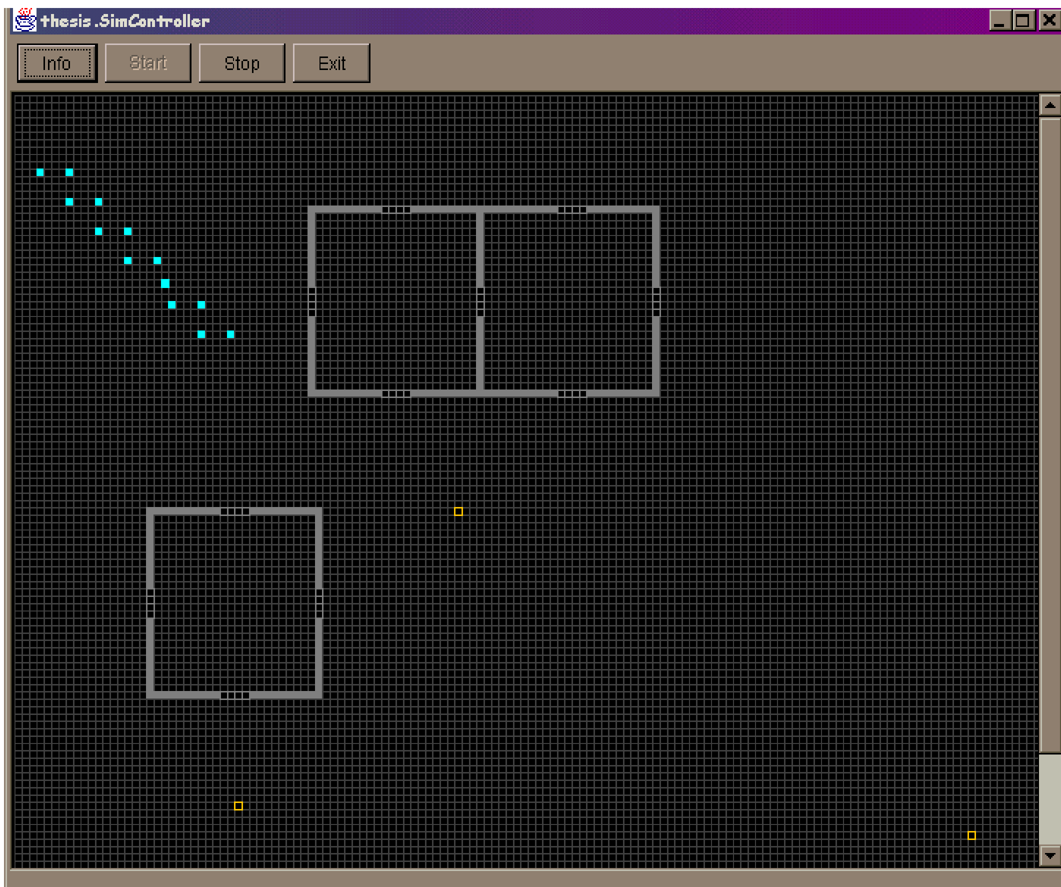


Figure 4. The View Component's Display.

3. Controller

The controller component controls the running of the application. It does so through the *SimController* class, which extends the *Applet* class. The *SimController* class has three main functions. The first function is to maintain a global "blackboard" of information by which the View and Model components can retrieve information about the simulation environment. The blackboard function is accomplished with an instantiation of the class *Parameters*. The *Parameters* class contains all of the simulation parameters and global constants of the simulation environment. It provides the common language for all 3 components of the simulation architecture.

The second function of the *SimController* class is to retrieve user input from external devices, i.e. mouse, keyboard, etc. This input is processed and translated into commands. These commands are then sent to the Model and View components in order to trigger their state changes.

The third function of the *SimController* class is to run the simulation iteration by iteration. In the case of the agents, the controller component will act as the system "referee" as the agents interact with each other and the environment. The resulting architecture provides for extensibility and modularity. In addition, it facilitates rapid prototyping of the system that allows functionality to be added incrementally.

Earlier attempts to use an independent thread to execute the simulation made it difficult to test and debug the program. To facilitate testability in MSCS, the controller uses a lockstep-timestep to run the simulation.

D. SIMULATION ENVIRONMENT

The *Parameters* class holds all the necessary information about the environment. It maintains a data structure *SIM_ENV* (short for *Simulation Environment*), which knows what type of *SimObject* occupies each grid square. *SIM_ENV* is updated after each simulation iteration to keep up with agent movement. The information provided by

SIM_ENV is critical for collision detection and avoidance in connection with agent movement during the simulation. This issue is discussed further in Chapter IV.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. AGENT DEVELOPMENT

A. AGENT FRAMEWORK

One of the key design issues of this program is that of the agent. Autonomous agents take input from their environment, process that input, and then execute actions that in turn affect the environment, (Weiss, 1999). Like an object in object-oriented programming, an agent is acted upon by its environment. However, unlike an object, the agent has sole direct control over its behavior and actions. As an object encapsulates its state and behaviors, so too will the agent's state and behaviors be encapsulated, along with its data processing and decision-making functions.

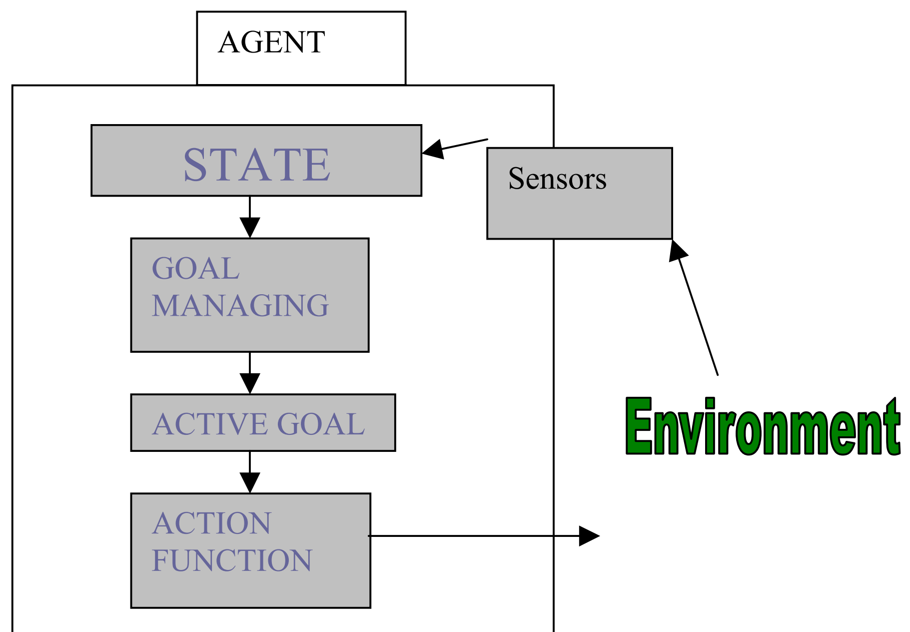


Figure 5. Agent Framework.

Figure 5 graphically depicts the high-level decision-cycle framework for the Marine agent or OODA loop. The agent *observes* the environment through its sensor

functions. The information received may or may not affect the agent's state. If the state has changed, the agent's new state will affect the status of its individual goals. The agent *orients* itself using its own state revision function to adjust its state variables. The goal managing mechanism of the agent will then adjust the status of its goals based on the agent's new state. The agent then *decides* using its goal managing mechanism which goal is the most critical and should be addressed. The goal manager assigns this goal as the active goal. Finally, the agent *acts* on the environment based on its active goal. The agent executes this decision cycle each iteration of the simulation.

B. STATE VARIABLES

Referring back to Figure 2, the *SimpleAgent* has an *X* and *Y* position, *type*, *heading*, and *health* attributes. An instantiation of the *Marine* agent contains these attributes and several more, the most important of which are the *role* attribute, the *rate* attribute and *activeGoal* attribute.

The *role* attribute is assigned to each agent at the moment of instantiation. It defines the Marine agent's job in the squad. It affects the actions of the agent and serves as a heuristic for its movement algorithm. Currently, the *role* attribute is not explored much by our implementation for lack of time. For future enhancement of the program changing the *role* attribute of an agent can help model delegation of authority if a senior agent is destroyed.

The *rate* attribute describes the rate of movement for the agent. Currently, in the *Parameters* class, there are 5 rates of movement defined as distances in pixel length per time step that the agent can to move. This translates to a range of movement of 0 to 2 grid squares per time step.

The *activeGoal* attribute is the goal that the agent deems critical to address. The *activeGoal* dictates the action that the agent will execute at the end of its decision cycle. The agent's goal-managing mechanism, a derivative of the *GoalManager* class described later in the chapter, determines this

C. GOALS AND GOAL MANAGING

The agent's behavior is dictated by its goals represented by Goal objects, which describe a desired state the agent works toward. At this stage of the simulation development, the Goal objects created for Marine agents are MoveGoal, FormationGoal, and the SurvivalGoal. MoveGoal means to move to a desired location in the simulation environment. FormationGoal means to assume a position in the squad formation based on its role attribute. The SurvivalGoal, not yet implemented, means to get out of the line of fire and find a place of cover to regenerate its health value.

Goal objects can hold one of three states. Using a traffic-light analogy, goals that become critical have a *RED* status; those goals that are satisfied hold a *GREEN* status; otherwise goals are in a *YELLOW* status. In the cases of the *MoveGoal* and *FormationGoal*, the current distance between the agent and its desired destination determines the status thresholds. For example, if the distance from the agent to its desired MoveGoal destination is greater than 100, then the agent's move goal would be assigned a *RED* status; if the distance is less than 5, the MoveGoal status would be *GREEN*; and if the distance is between 5 and 100, the MoveGoal status is *YELLOW*. The assessment of the *FormationGoal* status is similar, except it is based on the agent's distance from its current position to the desired location with respect to the squad leader position and heading. This ensures that the agents move in some semblance of a tactical formation. The assessment of the *SurvivalGoal* would be based on the *health* attribute of the agent or whether the agent is under fire.

The agent's overall goal managing is handled by a software mechanism simply called the *GoalManager* class. The *MarineGoalManager,* which extends the GoalManager class, holds the agent's *Goal* objects, assesses the status of these *Goal* objects after each state change, and then determines which goal is the most critical and should be addressed by the agent. It is in the *MarineGoalManager* that the priorities and decision-making aspects of the agent are created. For example, after assessing the status of its goals, the MarineGoalManager can assign the active goal by this algorithm:

IF *SurvivalGoal.status* = *RED* , THEN *activeGoal* = *SurvivalGoal*

ELSE IF *FormationGoal.status* = *RED*, THEN *activeGoa*l= *FormationGoal*

ELSE *activeGoal* = *MoveGoal.*

This algorithm would make survival the agent's highest priority, followed by staying in formation, and then moving towards the agent's final destination. This is a general solution for an average agent. There are other ways to give each agent a unique set of decision-making rules based on personality attributes like *experience*, *leadership*, *bravery*, etc. For example, some agents could hold the *MoveGoal* as a higher priority than staying in formation, or other agents could hold the *MoveGoal* as the highest priority overall. Similarly, varying the goal-status thresholds could create cautious agents whose *SurvivalGoal* becomes critical when their *health* attribute is 60 or less, aggressive agents whose *SurvivalGoal* becomes critical when their *health* attribute is 40 or less, or "kamikaze" (suicide) agents whose *SurvivalGoal* never becomes critical.

D. AGENT ACTION

1. Basic Movement

Once the agent's active goal has been assigned, the agent executes an action that will advance the agent further in the accomplishment of that goal. Presently, the agents are restricted to one type of action: movement. The agents will move toward a desired checkpoint or a position in the squad formation. Figure 6 depicts the agent's movement constraints. The grid layout of the simulation environment constrains agent movement to 8 possible directions shown by the arrows. The aspect of movement rate constrains the agent to move 0, 1 or 2 grid squares in any of the 8 directions. The squares in yellow represent movement at the fastest rate; the squares in green represent movement at the slow and medium rates; and the blue square represents the agent's position.

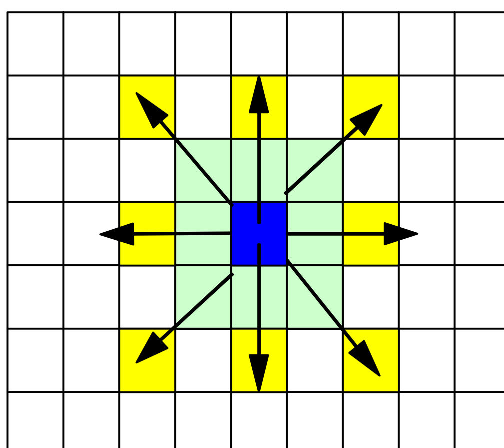


Figure 6. Agent Movement Constraints.

2 Collision Detection/Avoidance

Which of the 8 directions is the best for the agent to head in to reach a desired location is a straightforward evaluation. However, collision detection and avoidance become the critical issues for agent movement in the modeled urban infrastructure. The individual agent's movement is confined to 16 possible moves, so a simple and inexpensive algorithm was developed to check the viability of each of these moves by retrieving information about the corresponding grid squares from the SIM_ENV array. Knowing the agent's rate and preferred direction of travel, the agent's list of moves is cut down to 4, the last one being to stay in place. These moves are ordered by distance to the desired location. For example, if the agent needed to move down and right, but was unable to, his next set of moves would be to move down or move right. If neither is viable then the agent stays put for that time-step.

Although this collision detection algorithm works at the local level, it can fail at the global level. It has the tendency to get agents stuck in corners or areas with many obstacles. It also limits the agent movement to direct routes to the desired location, which is not always the smartest approach by human standards. These problems led to an attempt to give the agent some global obstacle sensing by a density evaluating function. In this context density describes the presence of obstacles and their proximity to the

agent. For a specified number of grid squares in each of the possible directions of movement, an integer value representing the type of object that occupies the grid square was retrieved from the SIM_ENV array. Values other than zero represent obstacles. Each value was then multiplied by a proximity factor; the closer the obstacle was to the agent, the higher the proximity factor. The results were summed together for a total density value for that direction. However, this approach still failed to give the agent a reliably viable route to its final destination: Agents were still getting stuck in corners or inside buildings. This led to the decision to incorporate path planning, at least for the squad leader.

3. Path Planning

Path planning gives the squad leader global planning ability to find a truly viable path from the starting point to the squad's desired destination. Only the agent whose *role* attribute was that of the squad leader would be given the ability to plan a path. This was done to visualize the difference in mission between the squad leader and the rest of the squad.

The cornerstone of path planning is a search algorithm. The search algorithm would need a search space of nodes to serve as a search graph. The nodes would have to correspond to trafficable areas in the environment. One simplification was to associate nodes with each entrance of the buildings in the environment and then connect the nodes that are within a clear line of sight of each other. The first step was to create a class of nodes called *PathNode* and a search-graph class, *PathSearchGraph*, to hold the nodes. Next, the Door class was reengineered to create two objects of the *PathNode* class upon instantiation of a *Door* object, one corresponding to the external side of the *Door* object and one corresponding to the internal side. These nodes are added to the *PathSearchGraph*, which then calls on a method to link all the nodes that are within a clear line of sight of each other. This creates the network of nodes. When the squad leader agent begins his path planning, he creates two additional *PathNodes*, corresponding to his starting position and his desired destination, and links them to the existing search graph.

Figure 7 shows an example of the search graph created from a sample infrastructure (wall and doors). The *PathNodes* are shown as the white colored squares associated with the doors of the gray buildings. The red lines are the edges that connect the nodes by line of sight. The squad members are the cyan squares.

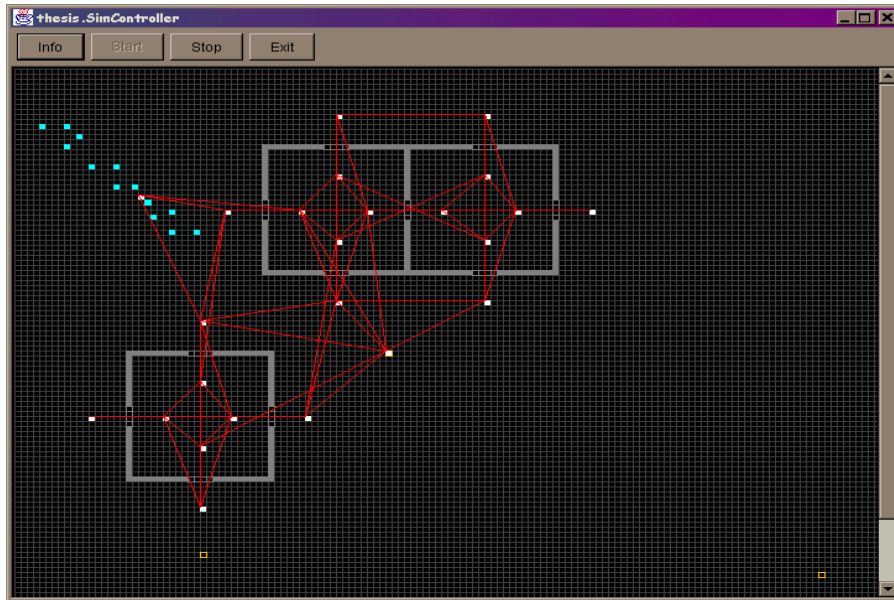


Figure 7. Path Search Node Network.

Once there is a network of nodes including the start and destination nodes, the squad leader agent can use one of four algorithms to choose a path to the destination node. Breadth-first and Depth-first search algorithms were programmed for this purpose; however, they are not practical as tactical path planning algorithms.

So, two best-first search algorithms were created. The first was based on distance from the node to the destination node. The algorithm finds the path of shortest distance from the start node to the destination node. The second best-first search algorithm was based on a cover and concealment benefit instead of a cost; nodes internal to the buildings were given a cover and concealment value of 10, and all others were given a cover and concealment value of 0. The algorithm searches for the most covered and concealed route to the destination node.

Once the search algorithm is able to locate the destination node a list of nodes from the start node to the destination node is compiled. Next, a path is constructed for each successive pair of nodes, providing a list of sub-destinations that the agent moves toward in order to reach its final destination. Figures 8 and 9 show the resulting paths (yellow squares) for the best-first search shortest path algorithm and the best-first search cover and concealment algorithm, respectively.

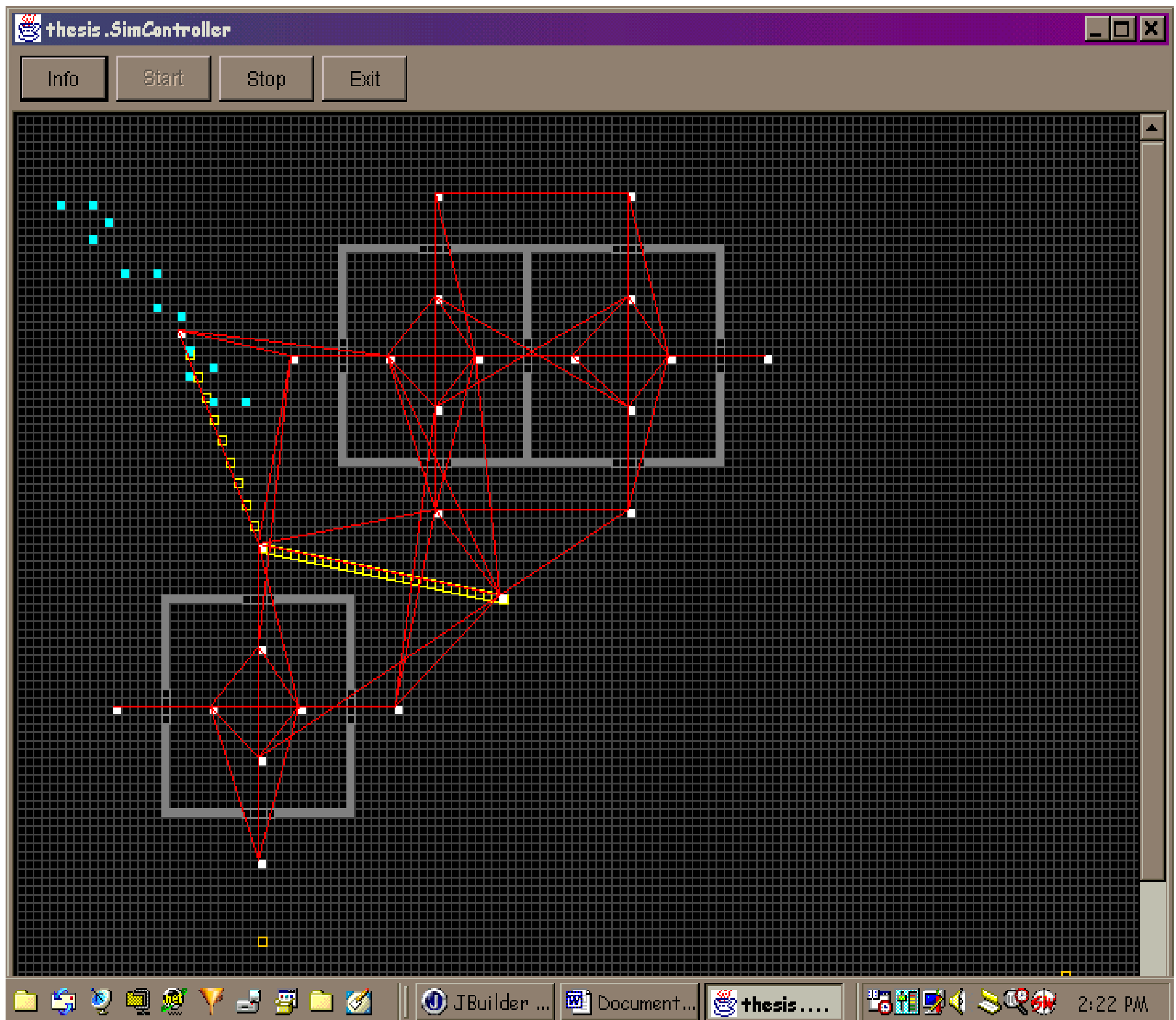


Figure 8. Best-First Search (Shortest Path).

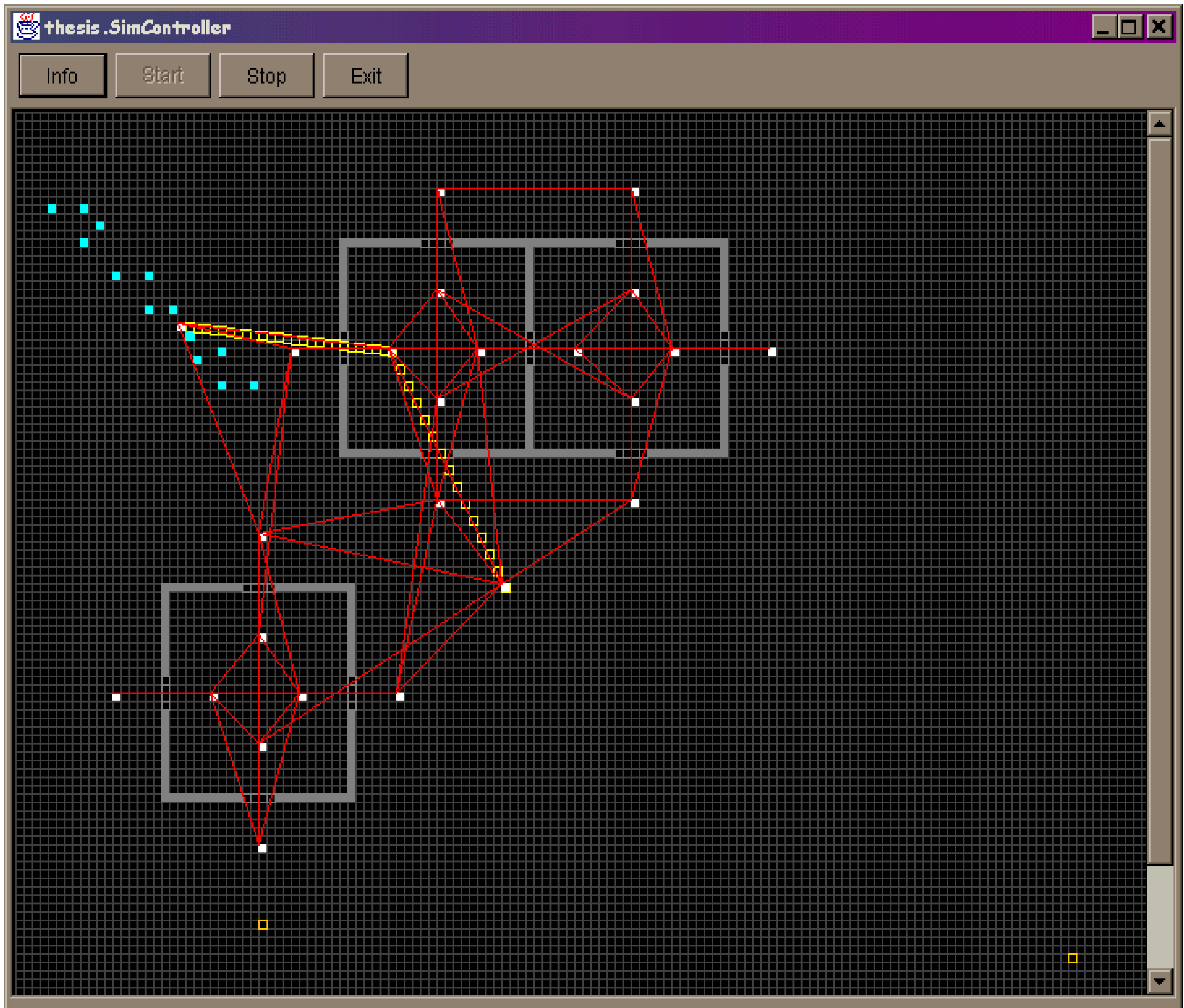


Figure 9. Best-First Search (Cover and Concealment Benefit).

Path planning solved the collision detection/avoidance problem and provided a way of modeling the common tactical dilemma of speed vs. security. In some situations a leader will choose to take the quickest or shortest route. In other situations, it is more important to find a secure route, characterized by cover and concealment. This and the variety of search algorithms provide ways to create unique tactical decision-making personalities.

E. SQUAD RELATIONSHIP

The relationship between the squad and its individual agents is a major issue in the development of this simulation. The intent was to engineer a software structure that could act as an agent while at the same time serve as the cohesive container for the individual *Marine* agents. The *Squad* entity would have to be transparent to the simulation environment. The *Squad* object, therefore, would not be derived from the *SimpleAgent* class, but would contain the same framework as the *Marine* agent.

The *Squad* entity would sense the environment and contain its own goals and goal manager like the individual agents. Its state would be dependent on the status of its goals and the states of the agents in the squad. It would use the personality attributes of the *Marine* whose *role* attribute is *Squad Leader* to influence its decision-making process. However, its main function would be to keep the agents focused on the squad's overall goal. Figure 10 shows the class relationship between the *Squad* agent and the Marine agents.

Figure 10. Squad-Marine Agents' Class Relationship.

F. SQUAD VS. INDIVIDUAL GOALS

Presently, the Squad manages only one goal: to conduct a patrol using predetermined checkpoints. This *PatrolGoal* directs the individual agents to move through these checkpoints by catalyzing the creation of successive *MoveGoal*s for the squad leader agent. The rest of the agents in the squad have highest priority on their *FormationGoal*, which directs the agents to maintain their position in the squad formation. The *SquadGoalManager* assesses the progress of the *PatrolGoal* and the simulation ends when the *PatrolGoal* status is *GREEN*.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION ANALYSIS

A RUNNING THE SIMULATION

Opening the *SimController* applet instantiates the simulation world and all the models in the world. The infrastructure is created. The squad and its agents are created and positioned at a predetermined starting point. The simulation advances each time-step with each click of the mouse button. With each time-step of the simulation, the agents in the squad execute their decision cycle with the ultimate intent of accomplishing the squad's overall goal and their individual goals. The simulation as a whole is comprised of 22 Java classes and the SimController.html applet. The program runs using the Java Development Kit version 1.3, or using the JBuilder 4 IDE and occupies 70 kilobytes of source data.

B. SIMULATION RESULTS

1. The Scenario

At this stage of development, the simulation program models an infantry squad containing 13 autonomous *Marine* agents. The squad has a single goal: to patrol the simulation environment, using predetermined checkpoints. There were three checkpoints in the scenario. The patrol mission is translated to the squad leader agent as successive movement goals. The squad leader agent receives the first destination from the *Squad* entity and creates a movement goal to that destination. Upon reaching the first checkpoint, the squad leader receives the second checkpoint and again creates a movement goal to that destination. This cycle repeats until the third checkpoint has been translated to the squad leader agent.

2. Squad and Individual Agent Interaction

The squad leader agent uses its path-planning ability to find a viable route to each checkpoint and then moves along the resulting path. The other agents in the squad use a local collision detection/avoidance algorithm to pursue their *FormationGoal*, essentially

leaving the navigation responsibilities to the squad leader. Their main concern is to maintain a column formation as they move from checkpoint to checkpoint. This was done to model a command and control structure within the squad. The *Squad* entity keeps track of the all the agents in the squad in order to assess the progress of meeting the *Squad*'s PatrolGoal.

The results on this simple scenario were mixed. The process of getting the squad through all three checkpoints was successful. The squad leader was able to move unhindered, while the rest of the squad guided their movement off of the squad leader. The majority of the squad reaches each checkpoint as a cohesive unit. However, members of the squad seem to act like "lost sheep" following a shepherd because only the shepherd knows where to go. In fact, because of the limitations in the collision detection and avoidance algorithm and lack of global planning strategies in the *Marine* agents, some ended getting stuck. It was common to see 1 or 2 agents getting stuck in corners while the rest of the squad continued on to the next checkpoint. Figure 11 shows the squad moving to the third and final checkpoint.

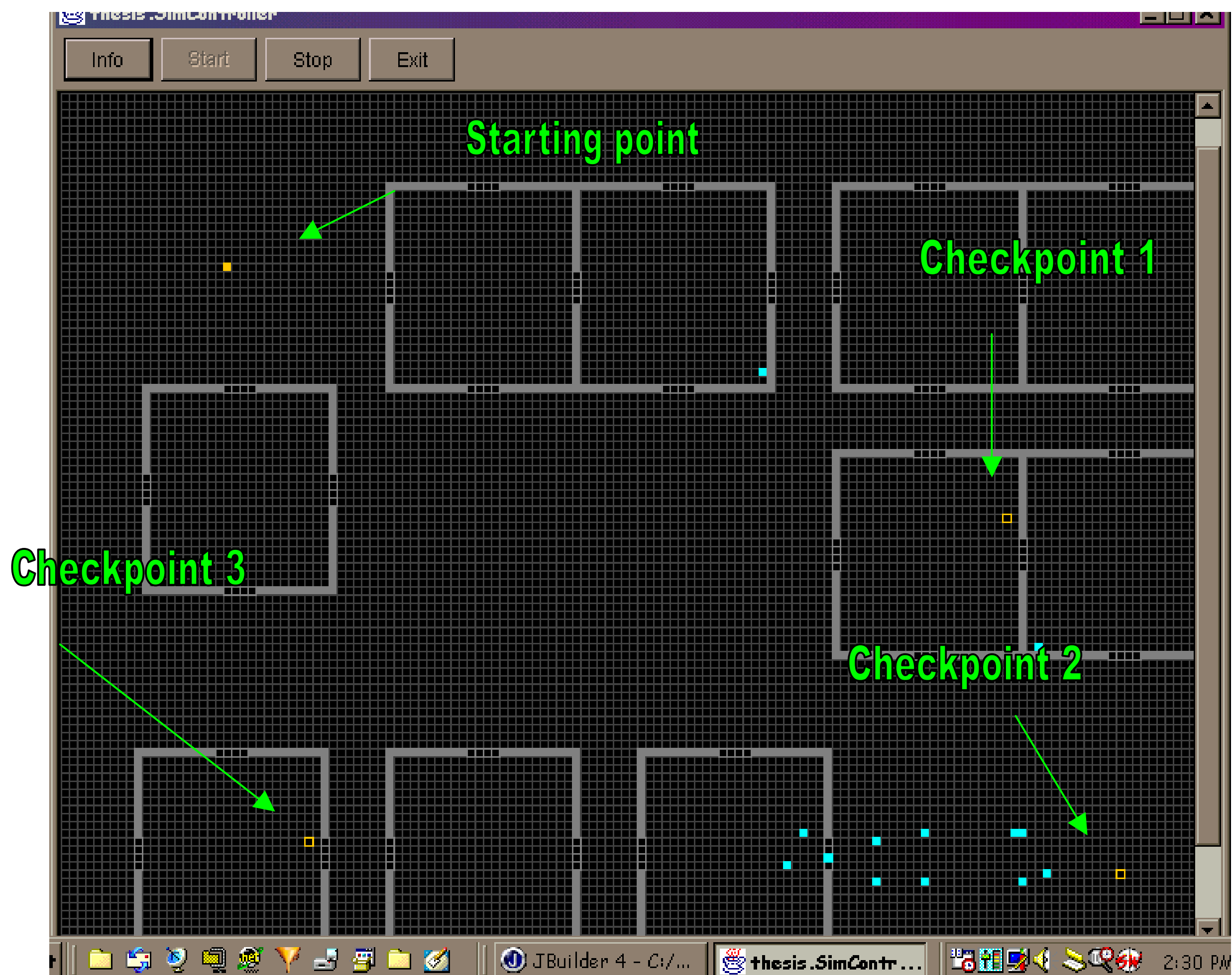


Figure 11. Squad Moving to Third Checkpoint.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. FUTURE WORK AND CONCLUSIONS

A. FUTURE WORK

Many potential enhancements of this program are apparent.

1. Develop Agent Personalities

- Develop the agents' personalities by adding tangible skills like marksmanship and physical fitness which would affect their shooting accuracy and movement rates.
- Add intangible qualities like experience, leadership, and courage, which would affect the agents' decision-making ability and goal prioritization.
- Refine the agent models by studying actual human performance.

2. Develop Agent Actions

- Add randomness to action choice and execution.
- Develop the agents' ability to act by adding target acquisition ability.
- Add the ability to engage enemy targets.
- Allow agents to die.

3. Develop different agent types

- Introduce enemy agents with opposing goals and non-combatant agents.

4. Develop the infrastructure

- Make the simulation terrain affect the agents' movement rates.
- Add more complexity to the infrastructure, water supplies, religious centers, etc, with cause-and-effect relationships with how the enemy and non-combatants act when these are affected by Marine agent actions.

5. Develop the User Interface

Add functionality to the graphical user interface in order to allow the user to:

- Change the simulation parameters;
- Alter the personalities of the agents;
- Analyze the current state of the agents;
- Add agents dynamically; and
- Analyze elements of the infrastructure.

6. Create Smarter Agents

Develop the intelligence of the agents in the simulation by:

- Creating more goals for the individual agent.
- Creating more goals for the squad.
- Using genetic algorithms to allow the agents to adapt and learn from their environment.

B. CONCLUSIONS

Developing autonomous agents is a complex and evolutionary undertaking. Detailed software engineering preparation is needed to produce intelligent mechanisms within the agent. The potential of autonomous agents in research and analysis in a simulation program remains promising for military applications. They provide a controllable and inexpensive test environment for a wide variety of processes. Using such a versatile tool to evaluate small-unit operations in an urban environment is an advantage over our future adversaries that we cannot afford to ignore.

APPENDIX A. JAVA SOURCE CODE

This appendix contains the Java source code for the Marine Squad Combat Simulation.

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.MenuBar;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;



public class SimController extends Applet implements ActionListener {
  boolean isStandalone = false;


  //simulation attributes
  int iteration=0;
  Squad simSquad;
  Marine tempMarine;
  Vector targetVector;
  Infrastructure simInfrastructure;
  Parameters simParameters;



  //graphics
  Image offScreenImage = null;
  Graphics offScreenGraphics = null;
  Dimension offScreenSize = null;
```

```
//interface components
BorderLayout borderLayout1;
Panel controlPanel;
Choice searchAlgorithm;
String algorithms[] = {"BestFirst (Cover & Concealment)",
    "BestFirst(Shortest Distance)", "BreadthFirstSearch",
    "DepthFirstSearch"};
Button simStep, reset, showNode, showPath;



/**Get a parameter value*/
public String getParameter(String key, String def) {
  return isStandalone ? System.getProperty(key, def) :
    (getParameter(key) != null ? getParameter(key) : def);
}


/**Construct the applet*/
public SimController() {

   addMouseListener( new MouseAdapter(){
       public void mouseReleased( MouseEvent me){
          simSquad.ooda();
          repaint();
          Parameters.simIteration++;

       }

   });

}


/**Initialize the applet*/
public void init() {
  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
/**Component initialization*/
private void jbInit() throws Exception {
    this.setBackground(Color.lightGray);
    borderLayout1 = new BorderLayout();
    this.setLayout(borderLayout1);

    controlPanel = new Panel();
    searchAlgorithm = new Choice();
    for (int ik = 0; ik < algorithms.length; ik++){
      searchAlgorithm.addItem(algorithms[ik]);
    }
    searchAlgorithm.addItemListener(
        new ItemListener(){
```

```
            public void itemStateChanged(ItemEvent e)
            {
              String mystring = (String)e.getItem();
              if("BreadthFirstSearch".equals(mystring)){
                 Parameters.PathSearch = Parameters.BFSEARCH;
              }
              else if("DepthFirstSearch".equals(mystring)){
                Parameters.PathSearch = Parameters.DFSEARCH;
              }
              else{
                if("BestFirst(Shortest Distance)".equals(mystring)){
                  Parameters.PathSearch = Parameters.SHORTDIST_SEARCH;
                }
                else if (
                    "BestFirst (Cover & Concealment)".equals(mystring)){
                    Parameters.PathSearch = Parameters.COVER_SEARCH;
                }
              }
            }
          });
      simStep = new Button("STEP");
      reset = new Button("RESET");
      showNode = new Button("SHOW NODES");
      showPath = new Button("SHOW PATH");

      controlPanel.setLayout(new GridLayout(1,5));
      controlPanel.add(simStep);
      simStep.addActionListener(this);
      controlPanel.add(reset);
      reset.addActionListener(this);
      controlPanel.add(showNode);
      showNode.addActionListener(this);
      controlPanel.add(showPath);
      showPath.addActionListener(this);
      controlPanel.add(searchAlgorithm);

      simParameters = new Parameters();
      simParameters.initEnv();
      simInfrastructure = new Infrastructure();
      simSquad = new Squad();
      simParameters.pathGraph.setLinks();
      this.add(controlPanel, borderLayout1.NORTH);


  }//end init()



  public void actionPerformed(ActionEvent e){
    String arg = e.getActionCommand();

    if("STEP".equals(arg)){
       startStop();
    }
    else if("RESET".equals(arg)){
      reset();
```

```
    }
    else{
      if("SHOW NODES".equals(arg)){
        toggleNodes();
      }
      else if("SHOW PATH".equals(arg)){
        togglePath();
      }

    }

}//end actionPerformed

public void startStop(){
    simSquad.ooda();
    repaint();
    Parameters.simIteration++;
}

public void reset(){
    simParameters = new Parameters();
    simParameters.initEnv();
    simInfrastructure = new Infrastructure();
    simSquad = new Squad();
    simParameters.pathGraph.setLinks();
    Parameters.simIteration = 0;
    repaint();
}

public void toggleNodes(){
     if(!Parameters.ShowNodes){
       Parameters.ShowNodes = true;
     }
     else{
       Parameters.ShowNodes = false;
     }
     repaint();
}//end toggleNodes


public void togglePath(){
  if(!Parameters.ShowPath){
     Parameters.ShowPath = true;
  }
  else{
     Parameters.ShowPath = false;
  }
   repaint();
}//end togglePath



public void paint(Graphics g){

    //draw background world
     g.setColor(Color.black);
```

```
    g.fillRect(0,0,Parameters.WIDTH, Parameters.HEIGHT);
    g.setColor(Color.darkGray);
    for(int ix =0; ix <= Parameters.COLUMNS; ix++){
        g.drawLine(ix*Parameters.CELL_WIDTH, 0,
        ix*Parameters.CELL_WIDTH,Parameters.HEIGHT);
    }
    for(int iy=0; iy<= Parameters.ROWS; iy++){
      g.drawLine(0,iy*Parameters.CELL_HEIGHT,
      Parameters.WIDTH, iy*Parameters.CELL_HEIGHT);
    }

    //draw start pt
    g.setColor(Color.orange);
    g.fillRect(Parameters.STARTING_POINT[0],Parameters.STARTING_POINT[1],
            Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);

    //draw checkPoints
    g.fillRect(Parameters.CHECK_PTS[0][0],Parameters.CHECK_PTS[0][1],
            Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
    g.fillRect(Parameters.CHECK_PTS[1][0],Parameters.CHECK_PTS[1][1],
            Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
    g.fillRect(Parameters.CHECK_PTS[2][0],Parameters.CHECK_PTS[2][1],
            Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);

   //draw infrastructure
   simInfrastructure.paint(g);

    //draw squad
    simSquad.paint(g);

    //draw nodes
    if(Parameters.ShowNodes){
     Parameters.pathGraph.paint(g);
    }

}//end paint


 public final void update(Graphics g){

   //implements no-flicker graphics
   Dimension dim = getSize();

   if((offScreenImage==null) ||
     (dim.width != offScreenSize.width) ||
     (dim.height != offScreenSize.height)){

     offScreenImage = createImage(dim.width, dim.height);
     offScreenSize = dim;
     offScreenGraphics = offScreenImage.getGraphics();

     }
   offScreenGraphics.clearRect(0,0,offScreenSize.width,offScreenSize.height);

   paint(offScreenGraphics);
   g.drawImage(offScreenImage,0,0,null);
```

```
  }//end update


  /**Start the applet*/
  public void start() {
  }
  /**Stop the applet*/
  public void stop() {
  }
  /**Destroy the applet*/
  public void destroy() {
  }
  /**Get Applet information*/
  public String getAppletInfo() {
    return "Applet Information";
  }
  /**Get parameter info*/
  public String[][] getParameterInfo() {
    return null;
  }
}
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *             in an urban environment
 * Copyright:   Copyright (c) 2001
 * Company:     USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

/**
 * class Parameters holds all the simulation parameters and
 * constants.
 */
public class Parameters {

  //simulation environment
  final static int CELL_WIDTH = 5;
  final static int CELL_HEIGHT= 5;
  final static int COLUMNS = 150;
  final static int ROWS = 120;
  final static int WIDTH = CELL_WIDTH * COLUMNS;
  final static int HEIGHT = CELL_HEIGHT * ROWS;

  //environment array
  //tracks what object occupies a grid in the environment
   public static int SIM_ENV[][] = new int[120][150];


   //simObject types
  final static int DOOR = -1;
  final static int MARINE =5;
  final static int WALL = 15;

   //simulation parameters
  final static int NUM_MARINES = 13;
  public static boolean ShowPath = false;
  public static boolean ShowNodes = false;
  public static int PathSearch = 0;
  public static boolean PatrolMission = false;

  //goal status
  final static int RED = -1;
  final static int YELLOW = 0;
  final static int GREEN = 1;

  final static int [] STARTING_POINT = {80,140};
  final static int [][] CHECK_PTS = {{520,280},{650,500},{100,480}};

  //rates of movement
  final static int STOP =0;
  final static int VERY_SLOW =1 ;
  final static int SLOW = 2;
  final static int MEDIUM = 5;
```

```
final static int FAST =10;

//simulation iteration counter
 public static int simIteration = 0;

//headings
final static int NORTH = 0;
final static int SOUTH = 1;
final static int EAST = 2;
final static int WEST = 3;
final static int NORTH_EAST = 4;
final static int NORTH_WEST = 5;
final static int SOUTH_EAST = 6;
final static int SOUTH_WEST = 7;

//move actions
final static int UP =0;
final static int DOWN =1;
final static int RIGHT = 2;
final static int LEFT =3;
final static int UPRIGHT =4;
final static int UPLEFT = 5;
final static int DOWNRIGHT = 6;
final static int DOWNLEFT = 7;
final static int STAY = 8;

 //pathsearchgraph
 public static PathSearchGraph pathGraph = new PathSearchGraph("Pathgraph");
 final static int LOS_DIST = 180; //Line of sight range
 final static int COVER_SEARCH = 0;//Bestfirst w/cover
 final static int SHORTDIST_SEARCH = 1;//=1 for depthfirst w/distance
 final static int BFSEARCH = 2;  //breadth first search
 final static int DFSEARCH = 3;//depth first search



//squad settings
public static int SQD_FORMATION = 2;
public static int DISPERSION = 2;

//formations
final static int LINE = 1;
final static int COLUMN = 2;

//goals
final static int MOVE_GOAL = 1;
final static int FORMATION_GOAL = 2;
final static int PATROL_GOAL = 3;

//squad roles
final static int SQD_LDR  = 0;

//wall sizes
final static int WALL_XS = 5;
final static int WALL_S = 10;
final static int WALL_M = 20;
```

```
 //door sizes
 final static int DOOR_S = 4;
 final static int DOOR_M = 6;
 //wall and door orientations
 final static int VERTICAL = 1;
 final static int HORIZONTAL = 2;


 /**
  * setEnv(r,c,t)
  * sets SIM_ENV[r][c] to type t
  */
 public void setEnv(int r, int c, int t){
     SIM_ENV[r][c] = t;
 }//end setEnv

/**
 * initEnv() initializes the SIM_ENV array
 */
 public void initEnv(){
   for(int ix = 0; ix < ROWS; ix++){
      for(int iy =0; iy < COLUMNS; iy++){
          SIM_ENV[ix][iy] = 0;
      }//end for
   }//end for
 }//end initEnv

 /**
  * getEnVal(r, c) returns type value contained
  * in SIM_ENV[r][c]
  */
 public int getEnVal(int r, int c){
   return SIM_ENV[r][c];
 }//end getEnVal

}//end Parameters
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

/**
 * class Squad creates an autonomous agent entity
 * that contains the Marine agents in the simulation
 * A transparent entity: has no discrete position in the
 * simulation environment.  Not a child of the SimObject class
 */

public class Squad {

  int direction = Parameters.RIGHT;
  int movementRate= Parameters.STOP;
  int squadActiveGoal = Parameters.PATROL_GOAL;
  Marine tempMarine;
  Vector squadVector;
  SquadGoalManager sqdGoalManager;



  public Squad() {

     squadVector = new Vector();

     //create marines and place them  in squad vector
     //position Marine agents in column formation on screen
     int baseX = Parameters.STARTING_POINT[0];
     int baseY =  Parameters.STARTING_POINT[1];
     int destH = Parameters.EAST;
     int destX = 0;
     int destY = 0;

     for(int ix = 0; ix < Parameters.NUM_MARINES; ix++){
        switch(ix){
          case 0:
             destX = baseX;
```

```
  destY = baseY;
break;
case 1:
 destX = baseX + (1 * Parameters.CELL_WIDTH);
 destY = baseY + (2 * Parameters.CELL_WIDTH);
break;

case 2:
 destX = baseX + (3 * Parameters.CELL_WIDTH);
 destY = baseY + (4 * Parameters.CELL_WIDTH);
break;

case 3:
 destX = baseX + (5 * Parameters.CELL_WIDTH);
 destY = baseY + (4 * Parameters.CELL_WIDTH);
break;

case 4:
  destX = baseX + (3 * Parameters.CELL_WIDTH);
  destY = baseY + (2 * Parameters.CELL_WIDTH);
break;

case 5:
  destX = baseX - (3 * Parameters.CELL_WIDTH);
  destY = baseY - (1 * Parameters.CELL_WIDTH);
break;

case 6:
   destX = baseX - (5 * Parameters.CELL_WIDTH);
   destY = baseY - (4 * Parameters.CELL_WIDTH);
break;

case 7:
   destX = baseX - (1* Parameters.CELL_HEIGHT);
   destY = baseY-(2 * Parameters.CELL_HEIGHT);

break;

case 8:
  destX = baseX -(3* Parameters.CELL_HEIGHT);
  destY = baseY-(4 * Parameters.CELL_HEIGHT);
break;

case 9:
 destX = baseX - (6 * Parameters.CELL_WIDTH);
 destY = baseY - (5 * Parameters.CELL_HEIGHT);
break;

case 10:
destX = baseX-(8 * Parameters.CELL_HEIGHT);
destY = baseY - (7 * Parameters.CELL_WIDTH);
break;

case 11:
 destX = baseX- (5 * Parameters.CELL_HEIGHT);
 destY = baseY - (5 * Parameters.CELL_WIDTH);
```

```
          break;

          case 12:
            destX = baseX - (6 * Parameters.CELL_WIDTH);
            destY = baseY - (7 * Parameters.CELL_HEIGHT);
          break;
        }//end switch(ix)

        //set Marine values
        tempMarine = new Marine(ix,destX,destY,destH);

        //add newly created Marine to squadVector
        squadVector.addElement(tempMarine);

      }//end for

      sqdGoalManager = new SquadGoalManager(squadVector);
  }//end constructor



/**
 * ooda() runs each Marine agent in the squad
 * through their ooda loop cycle
 */
  public void ooda(){
      sqdGoalManager.checkGoalStatus(squadVector);
      squadActiveGoal = sqdGoalManager.getActiveGoal();
      for(int ix = 0; ix < Parameters.NUM_MARINES; ix++){
        tempMarine = (Marine)squadVector.elementAt(ix);
        tempMarine.oodaLoop(squadVector);
      }
  }//end ooda


/**
 * paint() paints the Squad object
 */
   public void paint(Graphics g){
     for(int ix=0; ix < Parameters.NUM_MARINES; ix++){
         tempMarine = (Marine)squadVector.elementAt(ix);
         tempMarine.paint(g);

     }//end for
   }//end paint


/**
 * getMarineAt(x) returns Marine at index x
 */
  public Marine getMarineAt(int x){
     tempMarine = (Marine)squadVector.elementAt(x);
     return tempMarine;
  }//end getMarineAt

}//end squad
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

public class Marine extends SimpleAgent {

   //attributes
   int rate;
   int role;
   int personality[];
   int nextAgentMove = Parameters.STAY;
   int activeGoal;

   int SENSOR_RANGE = 10;
   int densityRange = 50;
   int density = 0;
   int densityThreshold = 25;
   int moveDensity[] = {0,0,0,0,0,0,0,0};
   int health;



   boolean[] moves ={true,true,true,true,true,true,true,true};
   Color agentColor = Color.blue;
   Marine tempMarine;
   Vector pathVector;
   PathElement tempElement;
   MarineGoalManager goalManager;



  public Marine(int r, int x, int y, int h) {
   setXpos(x);
   setYpos(y);
   setHeading(h);
   setType(Parameters.MARINE);
   Parameters.SIM_ENV[y/Parameters.CELL_HEIGHT]
             [x/Parameters.CELL_WIDTH]=Parameters.MARINE;
```

```
  role = r;
  rate = Parameters.SLOW;
  health = 100;
  pathVector = new Vector();
  goalManager = new MarineGoalManager(x,y,r,h,
                      Parameters.STARTING_POINT[0],
                      Parameters.STARTING_POINT[1]);
}//end constructor


//****************************methods***************************

/**
 * void setRate(int x)
 * sets Marines rate of movement
 */
public void setRate(int x){
   rate = x;
}

/**
 * void setRole(int x)
 * sets Marines role within the squad
 */
public void setRole(int x){
   role = x;
}

public void setNewMoveGoal(int x, int y){
  goalManager.newMoveGoal(x,y, this);
}//end setNewMoveGoal

public void setNewFormationGoal(int h,int bx,int by ){
  goalManager.newFormationGoal(h,bx,by);
}//end setNewFormationGoal

public void checkGoals(){
  goalManager.checkGoalStatus(getX(),
                           getY());
}//end checkGoals


public void paint(Graphics g){
   if(role == 0){
     g.setColor(Color.cyan);
     g.drawRect((getX()/Parameters.CELL_WIDTH)*Parameters.CELL_WIDTH,
      (getY()/Parameters.CELL_HEIGHT)*Parameters.CELL_HEIGHT,
           Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
      if(Parameters.ShowPath){
        g.setColor(Color.yellow);
       for(Enumeration e = pathVector.elements(); e.hasMoreElements();){
       tempElement = (PathElement)e.nextElement();
       g.drawRect((int)tempElement.getX(),
                (int)tempElement.getY(),
            Parameters.CELL_WIDTH, Parameters.CELL_HEIGHT);
       }
```

```
      }//end if


    }//end if

    g.setColor(Color.cyan);
    g.fillRect((getX()/Parameters.CELL_WIDTH)*Parameters.CELL_WIDTH,
      (getY()/Parameters.CELL_HEIGHT)*Parameters.CELL_HEIGHT,
           Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);


 }//end paint


public void oodaLoop(Vector sqdVector){
   tempMarine =(Marine)sqdVector.elementAt(0);
   int baseX = tempMarine.getX();
   int baseY = tempMarine.getY();
   int baseHeading = tempMarine.getHeading();
   if(role != 0){
    setNewFormationGoal(baseHeading,baseX,baseY);
   }


   goalManager.checkGoalStatus(getX(),getY());

   activeGoal = goalManager.getActiveGoal();


    switch(activeGoal){

      case Parameters.MOVE_GOAL:
        moveAction();
      break;

      case Parameters.FORMATION_GOAL:
        formationAction();
      break;

   }//end switch


}//end OODA loop


/**
  * int distance(cx,cy,gx,gy) returns the distance
  * between the points(cx,cy) and (gx,gy)
  * @return distance distance between two points
  */
public int distance(int cx, int cy, int gx, int gy){
   int distance=0;
   double delX = (cx-gx);
   double delY = (cy-gy);
   distance = (int)Math.sqrt(delX*delX + delY*delY);
   return distance;
```

```
}//end distance


/**
 * boolean lineOfSight(cx,cy,gx,gy) determines
 * if there is a clear line of sight between the two points
 * (cx,cy) and (gx,gy)
 * @return isLOS true if Line of Sight is clear
 */
public boolean lineOfSight(int cx, int cy, int gx, int gy){
  boolean isLOS = true;
  double delX = (cx-gx);
  double delY = (cy-gy);
  double iy;
  if(delX == 0){

      for( double ix = 5; ix <= Math.abs(delY); ix+=1){

      iy =  ix;
      if(delY<0){
         if(Parameters.SIM_ENV[(int)(cy +iy)/5][(int)cx/5]>10){
          isLOS = false;
         }//end if

      }
      else{
         if(Parameters.SIM_ENV[(int)(cy-iy)/5][(int)cx/5] >10){
            isLOS = false;
         }//end if

      }//end if/else

   }//end for


  }
  else{

     double slope = delY/delX;

     for( double ix = 5; ix <= Math.abs(delX); ix+=1){

      iy =  (slope*ix);
      if(delX<0){

         if(Parameters.SIM_ENV[(int)(cy+iy)/5][(int)(cx+ix)/5] >10){
            isLOS = false;
         }


      }
      else{

         if(Parameters.SIM_ENV[(int)(cy-iy)/5][(int)(cx-ix)/5] >10){
            isLOS = false;
         }
```

```
      }//end if/else

    }//end for

  }//endif/else

  return isLOS;

 }//lineOfSight


 /**
  * moveAction() executes the agent's action
  * to address the MoveGoal
  */
public void moveAction(){

    int goalX=0;
    int goalY=0;
    int currentX = getX();
    int currentY = getY();
    int distance = 0;
    int status= 0;
    goalX = goalManager.movegoal.getDestX();
    goalY = goalManager.movegoal.getDestY();
    distance = (int)Math.sqrt(Math.abs(goalX-currentX)*
             Math.abs(goalX-currentX) + Math.abs(goalY -currentY) *
           Math.abs(goalY -currentY));
    if(distance <= 2)rate = Parameters.STOP;
    if(distance < 2 && distance <=40) rate = Parameters.VERY_SLOW;
    if(distance >40 && distance <=100)rate = Parameters.SLOW;
    if(distance >100) rate = Parameters.MEDIUM;

   setMoves(currentX,currentY);
   setMoveDensity(currentX, currentY);

   //make first move in path
   Enumeration e = pathVector.elements();
   if(e.hasMoreElements()){
    tempElement = (PathElement) pathVector.elementAt(0);
    double pathX = tempElement.getX();
    double pathY = tempElement.getY();
    double deltaPos = Math.sqrt((pathX-currentX)*(pathX-currentX) +
                  (pathY-currentY)*(pathY-currentY));
      if(deltaPos <10){ pathVector.removeElementAt(0);}

    int statusX = currentX < pathX ? 1 : (currentX > pathX ? 3 : 2);
    int statusY = currentY < pathY ? 1 : (currentY > pathY ? 3 : 2);
     status = 10 * statusX + statusY;
    }
    else{

      status = 22;
    }
```

```
int nextMoveX= getX();
int nextMoveY = getY();

  switch(status){

      case 11:
           nextMoveX =getX()+rate;
           nextMoveY =getY()+rate;
           setHeading(Parameters.SOUTH_EAST);
           nextAgentMove = Parameters.SOUTH_EAST;
      break;

      case 12:
          nextMoveX = getX()+rate;
          nextMoveY = getY();
          setHeading(Parameters.EAST);
          nextAgentMove=Parameters.EAST;
      break;

       case 13:
           nextMoveX =getX()+rate;
           nextMoveY =getY()-rate;
           setHeading(Parameters.NORTH_EAST);
           nextAgentMove=Parameters.NORTH_EAST;
      break;

      case 21:
           nextMoveX = getX();
           nextMoveY = getY()+rate;
           setHeading(Parameters.SOUTH);
           nextAgentMove=Parameters.SOUTH;
      break;

      case 22:
           nextMoveX =getX();
           nextMoveY =getY();
           setHeading(getHeading());
           nextAgentMove=Parameters.STAY;
      break;

      case 23:
           nextMoveX = getX();
           nextMoveY = getY()-rate;
           setHeading(Parameters.NORTH);
           nextAgentMove=Parameters.NORTH;
      break;

      case 31:
          nextMoveX =getX()-rate;
          nextMoveY =getY()+rate;
          setHeading(Parameters.SOUTH_WEST);
          nextAgentMove=Parameters.SOUTH_WEST;
          break;

      case 32:
           nextMoveX = getX()-rate;
```

```
            nextMoveY = getY();
            setHeading(Parameters.WEST);
            nextAgentMove=Parameters.WEST;
        break;


        case 33:
            nextMoveX =getX()-rate;
            nextMoveY =getY()-rate;
            setHeading(Parameters.NORTH_WEST);
            nextAgentMove=Parameters.NORTH_WEST;
        break;


        default:
        break;

    }//end switch()

    setMoves(getX(),getY());
    if(nextAgentMove ==Parameters.STAY){

    }
    else{

       if(getX()/5 == nextMoveX/5 && getY()/5 == nextMoveY/5){
       //don't change SIM_Env
       }
       else{
        Parameters.SIM_ENV[getY()/5][getX()/5] = 0;
        Parameters.SIM_ENV[nextMoveY/5][nextMoveX/5] = Parameters.MARINE;
       }
        setXpos(nextMoveX);
        setYpos(nextMoveY);
    }
}//end action

/**
 * formationAction() executes the agent's action
 * to address its FormationGoal
 */
 public void formationAction(){
   int goalX=0;
   int goalY=0;
   int currentX = getX();
   int currentY = getY();
   int distance = 0;

   goalX = goalManager.formationgoal.getDestX();
   goalY = goalManager.formationgoal.getDestY();
   distance = (int)Math.sqrt(Math.abs(goalX-currentX)*
      Math.abs(goalX-currentX) + Math.abs(goalY -currentY) *
       Math.abs(goalY -currentY));
    if(distance <= 2) rate = Parameters.STOP;
    if (distance >2 && distance <=10) rate= Parameters.VERY_SLOW;
    if(distance >10 && distance <=20) rate = Parameters.SLOW;
```

```
  if(distance > 20 && distance <=30) rate = Parameters.MEDIUM;
  if(distance >30) rate = Parameters.FAST;

setMoves(currentX,currentY);
setMoveDensity(currentX, currentY);


int statusX = goalX > currentX ? 1 : (goalX < currentX ? 2 : 3);
int statusY = goalY > currentY ? 1 : (goalY < currentY ? 2 : 3);
int status = 10*statusX + statusY;

int ix = Parameters.STAY;
boolean loop = true;
nextAgentMove = Parameters.STAY;
switch(statusX){

   //agent should move right
    case 1:
        switch(statusY){

          //agent should move down
          case 1:
            nextAgentMove = Parameters.DOWNRIGHT;
           loop = true;
          while(loop && !moves[nextAgentMove] &&
               (moveDensity[nextAgentMove] > densityThreshold) ){
                switch(nextAgentMove){
                  case Parameters.DOWNRIGHT:
                  nextAgentMove = Parameters.RIGHT;
                  break;

                  case Parameters.RIGHT:
                  nextAgentMove = Parameters.DOWN;
                  break;

                  case Parameters.DOWN:
                  nextAgentMove = Parameters.DOWNLEFT;
                  break;

                  case Parameters.DOWNLEFT:
                  nextAgentMove = Parameters.UPRIGHT;
                  break;


                  case Parameters.UPRIGHT:
                  nextAgentMove = Parameters.STAY;
                  loop = false;
                  break;
               }//end switch(ix)
            }//end while loop
          break;


          //agent should move up
          case 2:
             ix = Parameters.UPRIGHT;
```

```
  loop = true;
  while(loop && !moves[ix] &&
        moveDensity[ix] > densityThreshold){
    switch(ix){
      case Parameters.UPRIGHT:
      ix = Parameters.RIGHT;
      break;

      case Parameters.RIGHT:
      ix = Parameters.UP;
      break;

      case Parameters.UP:
      ix = Parameters.UPLEFT;
      break;

      case Parameters.UPLEFT:
      ix = Parameters.DOWNRIGHT;
      break;

      case Parameters.DOWNRIGHT:
      loop = false;
      ix = Parameters.STAY;
      break;
    }//end switch(ix)
  }
  nextAgentMove = ix;
break;

//agent should MOVE right
case 3:
 ix = Parameters.RIGHT;
 loop = true;
  while(loop && !moves[ix] &&
        moveDensity[ix] > densityThreshold){
    switch(ix){
      case Parameters.RIGHT:
      ix = Parameters.UPRIGHT;
      break;

      case Parameters.UPRIGHT:
      ix = Parameters.DOWNRIGHT;
      break;

      case Parameters.DOWNRIGHT:
      ix = Parameters.UP;
      break;

      case Parameters.UP:
      ix = Parameters.DOWN;
      break;

      case Parameters.DOWN:
      loop = false;
      ix = Parameters.STAY;
      break;
```

```
          }//end switch(ix)
        }
        nextAgentMove = ix;
      break;



  }//end switch(statusY)
 break;


//agent should move left
 case 2:

   switch(statusY){

      //agent should move up
      case 2:
        ix = Parameters.UPLEFT;
        loop = true;
        while(loop &&!moves[ix] &&
           moveDensity[ix] > densityThreshold){
          switch(ix){
            case Parameters.UPLEFT:
            ix = Parameters.LEFT;
            break;

            case Parameters.LEFT:
            ix = Parameters.UP;
            break;

            case Parameters.UP:
            ix = Parameters.UPRIGHT;
            break;

            case Parameters.UPRIGHT:
            ix = Parameters.DOWNLEFT;
            break;

            case Parameters.DOWNLEFT:
            loop = false;
            ix = Parameters.STAY;
            break;
          }//end switch(ix)
        }//end while
        nextAgentMove = ix;
      break;

      //agent should move left
      case 3:
        ix = Parameters.LEFT;
        loop = true;
        while(loop && !moves[ix] &&
            moveDensity[ix] > densityThreshold){
          switch(ix){
            case Parameters.LEFT:
```

```
        ix = Parameters.UPLEFT;
        break;

        case Parameters.UPLEFT:
        ix = Parameters.DOWNLEFT;
        break;

        case Parameters.DOWNLEFT:
        ix = Parameters.UP;
        break;

        case Parameters.UP:
        ix = Parameters.DOWN;
        break;

        case Parameters.DOWN:
        loop = false;
        ix = Parameters.STAY;
        break;
      }//end switch(ix)
    }
    nextAgentMove = ix;
  break;

  //agent should move down
  case 1:
    ix = Parameters.DOWNLEFT;
    loop = true;
    while(loop && !moves[ix] &&
          moveDensity[ix] > densityThreshold){
      switch(ix){
        case Parameters.DOWNLEFT:
        ix = Parameters.LEFT;
        break;

        case Parameters.LEFT:
        ix = Parameters.DOWN;
        break;

        case Parameters.DOWN:
        ix = Parameters.UPLEFT;
        break;

        case Parameters.UPLEFT:
        ix = Parameters.DOWNRIGHT;
        break;

        case Parameters.DOWNRIGHT:
        loop = false;
        ix = Parameters.STAY;
        break;
      }//end switch(ix)
    }
    nextAgentMove = ix;
  break;
```

```
   }//end switch(statusY)

break;

//agent moves up or down
case 3:
   switch(statusY){

      //agent should move up
      case 2:
       ix = Parameters.UP;
       loop = true;
        while(loop && !moves[ix] &&
              moveDensity[ix] > densityThreshold){
          switch(ix){
            case Parameters.UP:
            ix = Parameters.UPLEFT;
            break;

            case Parameters.UPLEFT:
            ix = Parameters.UPRIGHT;
            break;

            case Parameters.UPRIGHT:
            ix = Parameters.LEFT;
            break;

            case Parameters.LEFT:
            ix = Parameters.RIGHT;
            break;

            case Parameters.RIGHT:
            loop = false;
            ix = Parameters.STAY;
            break;
          }//end switch(ix)
        }
        nextAgentMove = ix;
      break;
      //agent should stay
      case 3:
      nextAgentMove = Parameters.STAY;
      break;

      //agent should move down
      case 1:
      ix = Parameters.DOWN;
      loop = true;
        while(loop && !moves[ix] &&
               moveDensity[ix] > densityThreshold){
          switch(ix){
            case Parameters.DOWN:
            ix = Parameters.DOWNRIGHT;
            break;

            case Parameters.DOWNRIGHT:
```

```
                ix = Parameters.DOWNLEFT;
                break;

                case Parameters.DOWNLEFT:
                ix = Parameters.RIGHT;
                break;

                case Parameters.RIGHT:
                ix = Parameters.LEFT;
                break;

                case Parameters.LEFT:
                loop = false;
                ix = Parameters.STAY;
                break;
              }//end switch(ix)
            }
            nextAgentMove = ix;
          break;

        }//end switch(statusY)

    break;
  }//end switch(status)


  int nextAgentMoveX = getX();
  int nextAgentMoveY = getY();
    switch(nextAgentMove){

        case Parameters.STAY:
            nextAgentMoveX =getX();
            nextAgentMoveY =getY();
            super.setHeading(getHeading());
        break;

        case Parameters.UP:
            nextAgentMoveX = getX();
            nextAgentMoveY = getY()-rate;
            super.setHeading(Parameters.NORTH);
        break;

        case Parameters.DOWN:
            nextAgentMoveX = getX();
            nextAgentMoveY = getY()+rate;
            super.setHeading(Parameters.SOUTH);
        break;

        case Parameters.RIGHT:
           nextAgentMoveX = getX()+rate;
           nextAgentMoveY = getY();
           setHeading(Parameters.EAST);
        break;

        case Parameters.LEFT:
            nextAgentMoveX = getX()-rate;
```

```
            nextAgentMoveY = getY();
            super.setHeading(Parameters.WEST);
        break;

        case Parameters.UPRIGHT:
            nextAgentMoveX =getX()+rate;
            nextAgentMoveY =getY()-rate;
            super.setHeading(Parameters.NORTH_EAST);
        break;

        case Parameters.UPLEFT:
            nextAgentMoveX =getX()-rate;
            nextAgentMoveY =getY()-rate;
            super.setHeading(Parameters.NORTH_WEST);
        break;

        case Parameters.DOWNRIGHT:
            nextAgentMoveX =getX()+rate;
            nextAgentMoveY =getY()+rate;
            super.setHeading(Parameters.SOUTH_EAST);
        break;

        case Parameters.DOWNLEFT:
           nextAgentMoveX =getX()-rate;
           nextAgentMoveY =getY()+rate;
           super.setHeading(Parameters.SOUTH_WEST);
           break;

        default:
        break;

    }//end switch




    if(getX()/5 == nextAgentMoveX/5 && getY()/5 == nextAgentMoveY/5){
      //don't change SIM_Env
    }
    else{
     Parameters.SIM_ENV[getY()/5][getX()/5] = 0;
     Parameters.SIM_ENV[nextAgentMoveY/5][nextAgentMoveX/5] =
     Parameters.MARINE;
    }
    setXpos(nextAgentMoveX);
    setYpos(nextAgentMoveY);


}//end formationaction

/**
 * findPath(cx,cy,gx,gy) finds a path from point(cx,cy) to
 * (gx,gy)
 */
```

```
public void findPath(int cx, int cy, int gx, int gy){
 setMoves(cx,cy);
 PathElement newElement;
 double delX = (cx-gx);
 double delY = (cy-gy);
 double iy;
 if(delX == 0){
     for( double ix = 5; ix <= Math.abs(delY); ix+=5){

     iy =  ix;
     if(delY<0){
       if(Parameters.SIM_ENV[(int)(cy +iy)/5][(int)cx/5]<10){
         newElement = new PathElement((double)cx,(double)cy + iy);
         pathVector.addElement(newElement);
       }//end if
     }
     else{
       if(Parameters.SIM_ENV[(int)(cy-iy)/5][(int)cx/5] <10){
         newElement = new PathElement((double)cx,(double)cy - iy);
         pathVector.addElement(newElement);
       }//end if
     }//end if/else

   }//end for
  }
 else{

    double slope = delY/delX;

    for( double ix = 5; ix <= Math.abs(delX); ix+=5){

     iy =  (slope*ix);
     if(delX<0){

       if(Parameters.SIM_ENV[(int)(cy+iy)/5][(int)(cx+ix)/5] <10){
        newElement = new PathElement((double)cx + ix,(double)cy + iy);
        pathVector.addElement(newElement);
       }//end if

     }
     else{

       if(Parameters.SIM_ENV[(int)(cy-iy)/5][(int)(cx-ix)/5] <10){
         newElement = new PathElement((double)cx - ix,(double)cy - iy);
         pathVector.addElement(newElement);
        }//end if

     }//end if/else

   }//end for

 }//endif/else
}//end findPath()


/**
```

```
 * int getDensity() calculates the density
 * around the agent's current position
 */
public int getDensity(){

  //reset density value
  int newDensity = 0;

  int minX = getX()/Parameters.CELL_WIDTH - SENSOR_RANGE;
  minX = minX <=0 ? 0 : minX;

  int maxX = getX()/Parameters.CELL_HEIGHT + SENSOR_RANGE;
  maxX = maxX >= Parameters.COLUMNS - 1 ? Parameters.COLUMNS : maxX;

  int minY = getY()/Parameters.CELL_HEIGHT - SENSOR_RANGE;
  minY = minY<=0 ? 0 : minY;

  int maxY = getY()/Parameters.CELL_HEIGHT + SENSOR_RANGE;
  maxY = maxY >= Parameters.ROWS -1 ? Parameters.ROWS : maxY;

  for(int iy = minY; iy < maxY; iy++){
    for(int ix = minX; ix < maxX; ix++){
       newDensity += Parameters.SIM_ENV[iy][ix];
    }
  }

  return newDensity;

}//end getDensity



/**
 * setMoves(x,y) determines whether the 8
 * moves from point (x,y) are accessible,
 * i.e. no obstacles or other agents occupy the 8 grid squares
 * around the agent's current position
 */
public void setMoves(int x, int y){
 int currentX = x;
 int currentY = y;
 //reset moves
 for(int ix = 0; ix < moves.length; ix++){
    moves[ix] = true;
 }

 //check upmoves
   if(currentY <= 10 ||
         Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT - 1]
             [currentX/Parameters.CELL_WIDTH] > 10 ||
             (rate == Parameters.FAST &&
         Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT - 2]
             [currentX/Parameters.CELL_WIDTH] > 10 ))
       moves[Parameters.UP] = false;

   //checkdownMoves
```

```
if(currentY >= (Parameters.HEIGHT - 10)||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT + 1]
      [currentX/Parameters.CELL_WIDTH] > 10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT + 2]
      [currentX/Parameters.CELL_WIDTH] > 10 ))
    moves[Parameters.DOWN] = false;

//check leftmoves
if(currentX <= 10 ||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT]
      [currentX/Parameters.CELL_WIDTH - 1] > 10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT]
      [currentX/Parameters.CELL_WIDTH - 2] > 10))
    moves[Parameters.LEFT] = false;

//check rightmoves
if(currentX >= (Parameters.WIDTH - 10) ||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT]
      [currentX/Parameters.CELL_WIDTH + 1] >10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT]
      [currentX/Parameters.CELL_WIDTH + 2] >10 ))
    moves[Parameters.RIGHT] = false;

//check upright/downright/upleft/downleft moves
if(!moves[Parameters.UP] ||!moves[Parameters.RIGHT]||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT-1]
      [currentX/Parameters.CELL_WIDTH + 1] > 10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT-2]
      [currentX/Parameters.CELL_WIDTH + 2] > 10))
     moves[Parameters.UPRIGHT] = false;

if(!moves[Parameters.UP] ||!moves[Parameters.LEFT]||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT-1]
      [currentX/Parameters.CELL_WIDTH- 1] > 10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT-2]
      [currentX/Parameters.CELL_WIDTH- 2] > 10 ))
     moves[Parameters.UPLEFT] = false;

if(!moves[Parameters.DOWN] || !moves[Parameters.RIGHT]||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT+1]
      [currentX/Parameters.CELL_WIDTH + 1] > 10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT+2]
      [currentX/Parameters.CELL_WIDTH + 2] > 10))
     moves[Parameters.DOWNRIGHT] = false;

if(!moves[Parameters.DOWN] ||!moves[Parameters.LEFT]||
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT+1]
      [currentX/Parameters.CELL_WIDTH -1] > 10 ||
      (rate == Parameters.FAST &&
      Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT+2]
```

```
         [currentX/Parameters.CELL_WIDTH -2] > 10))
        moves[Parameters.DOWNLEFT] = false;

}//end setMoves()



/**
 * setMoveDensity(cx,cy) calculates the density
 * in each of the agent's directions of movement from its
 * position at (cx,cy) and places the value in the moveDensity array
 */
public void setMoveDensity(int cx, int cy){
  int currentX = cx;
  int currentY = cy;
  //reset moveDensity array
  for(int ik =0; ik < moveDensity.length; ik++){
    moveDensity[ik]=0;
  }
  int count = 0;
  int range = 0;
   //up density
   range = currentY/Parameters.CELL_HEIGHT - densityRange;
   range = range <= 0 ? 0 : range;
   for(int x = currentY/Parameters.CELL_HEIGHT - 1; x >= range; x--){
      count++;
      moveDensity[Parameters.UP] +=
            densityRange/count *
             Parameters.SIM_ENV[x][currentX/Parameters.CELL_WIDTH];
   }


   //down density
   count = 0;
   range = currentY/Parameters.CELL_HEIGHT +densityRange;
   range = range >= Parameters.ROWS - 1 ? Parameters.ROWS - 1 : range;
   for(int x = currentY/Parameters.CELL_HEIGHT +1; x <= range; x++){
      count++;
      moveDensity[Parameters.DOWN] +=
          densityRange/count *
          Parameters.SIM_ENV[x][currentX/Parameters.CELL_WIDTH];
   }

   //right density
   count = 0;
   range = currentX/Parameters.CELL_HEIGHT + densityRange;
   range = range >= Parameters.COLUMNS - 1 ? Parameters.COLUMNS - 1 : range;
   for(int x = currentX/Parameters.CELL_HEIGHT +1; x <= range; x++){
      count++;
      moveDensity[Parameters.RIGHT] +=
            densityRange/count *
             Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT][x];
   }


   //left density
```

```
 count =0;
 range = currentX/Parameters.CELL_HEIGHT - densityRange;
 range = range <= 0 ?  0 : range;
 for(int x = currentX/Parameters.CELL_HEIGHT -1; x >= range; x--){
    count++;
    moveDensity[Parameters.LEFT] +=
          densityRange/count *
          Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT][x];
 }

//up right
count = 0;
int upRange = currentY/Parameters.CELL_HEIGHT - densityRange;
 upRange = upRange <= 0 ?  0 : upRange;
int delUp = (int)Math.abs(currentY/Parameters.CELL_HEIGHT - upRange);

int rightRange=currentX/Parameters.CELL_HEIGHT + densityRange;
 rightRange = rightRange >= Parameters.COLUMNS - 1 ?
             Parameters.COLUMNS - 1 : rightRange;
int delRight = (int)Math.abs(currentX/Parameters.CELL_WIDTH - rightRange);

range = delUp < delRight ? delUp : delRight;

for(int x = 1; x < range; x++){
   count++;
   moveDensity[Parameters.UPRIGHT]+=
     densityRange/count *
     Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT - x]
                          [currentX/Parameters.CELL_WIDTH + x];
}


//up left
count =0;
 upRange = currentY/Parameters.CELL_HEIGHT - densityRange;
 upRange = upRange <= 0 ?  0 : upRange;
 delUp = (int)Math.abs(currentY/Parameters.CELL_HEIGHT - upRange);

int leftRange=currentX/Parameters.CELL_HEIGHT - densityRange;
leftRange =  leftRange <= 0 ? 0 : leftRange;

int delLeft = (int)Math.abs(currentX/Parameters.CELL_WIDTH - leftRange);
range = delUp < delLeft ? delUp : delLeft;

for(int x = 1; x < range; x++){
   count++;
   moveDensity[Parameters.UPLEFT]+=
       densityRange/count *
        Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT - x]
                          [currentX/Parameters.CELL_WIDTH - x];
}


//down right
count = 0;
int downRange = currentY/Parameters.CELL_HEIGHT + densityRange;
```

```
    downRange = downRange >= Parameters.ROWS -1 ?
                    Parameters.ROWS -1 : downRange;
    int delDown = (int)Math.abs(currentY/Parameters.CELL_HEIGHT - downRange);

    rightRange=currentX/Parameters.CELL_HEIGHT + densityRange;
    rightRange =  rightRange >= Parameters.COLUMNS - 1 ?
          Parameters.COLUMNS - 1 : rightRange;
    delRight = (int)Math.abs(currentX/Parameters.CELL_WIDTH - rightRange);

    range = delDown < delRight ? delDown : delRight;

    for(int x = 1; x < range; x++){
      count++;
      moveDensity[Parameters.DOWNRIGHT] +=
        densityRange/count *
         Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT + x]
                    [currentX/Parameters.CELL_WIDTH + x];
    }


    //down left
    count = 0;
    downRange = currentY/Parameters.CELL_HEIGHT + densityRange;
    downRange = downRange >= Parameters.ROWS -1 ?
                 Parameters.ROWS -1 : downRange;
    delDown = (int)Math.abs(currentY/Parameters.CELL_HEIGHT - downRange);

    leftRange=currentX/Parameters.CELL_HEIGHT - densityRange;
    leftRange = leftRange <= 0 ?  0 : leftRange;
    delLeft = (int)Math.abs(currentX/Parameters.CELL_WIDTH - leftRange);

    range = delDown < delLeft ? delDown : delLeft;

    for(int x = 1; x < range; x++){
      count++;
      moveDensity[Parameters.DOWNLEFT]+=
        densityRange/count *
        Parameters.SIM_ENV[currentY/Parameters.CELL_HEIGHT + x]
                    [currentX/Parameters.CELL_WIDTH - x];
    }//end for loop

  }//end setmoveDensity


}//end Marine
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

/**
 * class SimpleAgent
 * Description: child class of SimObject, parent class of all agents
 */

public class SimpleAgent extends SimObject {

  //attributes
  int heading;    //int between 0 and 359
  int health;     //value between 0 and 100


  public SimpleAgent() {
     health = 100;
     setType(0);
  }

  //set methods


  /**
   * void setHeading(int hd)
   * sets agent's heading to hd
   */
  public void setHeading(int hd){
     heading = hd;
  }

  /**
   * void setHealth(int hl)
   * sets agent's health to hl
   */
  public void setHealth(int hl){
     health = hl;
  }




  //get methods


  /**
   * int getHeading()
```

```
     * returns agent's heading
     */
    public int getHeading(){
        return heading;
    }

    /**
     * int getHealth()
     * returns agent's health
     */
    public int getHealth(){
        return health;
    }

}//end simpleAgent
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.util.*;

/**
 * SquadGoalManager manages the squads goal
 * Currently squad has a single goal: PatrolGoal
 */
public class SquadGoalManager  {
  PatrolGoal pgoal;


  public SquadGoalManager(Vector squadVector) {
    pgoal = new PatrolGoal(squadVector);
  }//

  public void checkGoalStatus(Vector squadVector){
    //check patrol goal status
    pgoal.checkGoalStatus(squadVector);
  }//end checkGoalStatus


  public int getActiveGoal(){
    int squadActiveGoal= Parameters.PATROL_GOAL;
    return squadActiveGoal;
  }//end getActiveGoal

}//end SquadGoalManager
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.util.*;

/**
 * MarineGoalManager manages the Marine agents' goals
 */

public class MarineGoalManager  {
  int agentRole;
  MoveGoal movegoal;
  FormationGoal formationgoal;

  /**
   * MarineGoalManager constructor creates
   * MoveGoal and FormationGoal objects
   * @param x move destination x-coord
   * @param y move destnation y-coord
   * @param r role of agent
   * @param h heading of squad leader
   * @param bX x-coord of base of movement(sqd leader)
   * @param bY y-coord of base of movement(sqd leader)
   */
  public MarineGoalManager(int x, int y, int r, int h, int bX, int bY) {

    if(r ==0){
       movegoal = new MoveGoal(x,y);
    }
    else{
      formationgoal = new FormationGoal(r,h,bX,bY);
    }
     agentRole = r;
  }//end constructor

  /**
   * newMoveGoal(int x, int y, Marine m)
   * creates a newMoveGoal for Marine m
   * for a destination of (x,y)
   * @param x x-coord of goal destination
   * @param y y-coord of goal destination
   * @param m Marine to receive to new move goal
   */
  public void newMoveGoal(int x, int y, Marine m){
     movegoal = new MoveGoal(x,y);
     PathNode startNode = new PathNode(m.getX(),m.getY(),0);
     PathNode endNode = new PathNode(x, y, 0);
```

```
  Parameters.pathGraph.reset();
  Parameters.pathGraph.put(startNode);
  Parameters.pathGraph.put(endNode);
  Parameters.pathGraph.setLinks();
  Vector waypoints = new Vector();
  switch(Parameters.PathSearch){

    case Parameters.COVER_SEARCH:
    waypoints = (Vector)Parameters.pathGraph.coverFirstSearch(startNode,
            endNode);
    break;

    case Parameters.SHORTDIST_SEARCH:
    waypoints = (Vector)Parameters.pathGraph.bestFirstSearch(startNode,
             endNode);
    break;

    case Parameters.BFSEARCH:
    waypoints = (Vector)Parameters.pathGraph.breadthFirstSearch(startNode,
             endNode);
    break;

    case Parameters.DFSEARCH:
    waypoints = (Vector)Parameters.pathGraph.depthFirstSearch(startNode,
             endNode);
    break;

  }//end switch

  for(int ij = waypoints.size()-1; ij > 0; ij--){
    PathNode tipNode = (PathNode)waypoints.elementAt(ij);
    PathNode tailNode = (PathNode)waypoints.elementAt(ij-1);

    m.findPath(tipNode.getX(),tipNode.getY(),
                     tailNode.getX(),tailNode.getY());
  }//end for

}//end newMoveGoal

/**
 * newFormationGoal(int h, int bX, int bY)
 * sets new FormationGoal
 * @param h squad leader's heading
 * @param bX squad leader's x-coord
 * @param bY squad leader's y-coord
 */
public void newFormationGoal(int h, int bX, int bY){
 formationgoal.setFormation(h,bX,bY);
}

/**
 * checkGoalStatus(int ax, int ay)
 * checks goal status based on the agent's current position
 * currently squad leader moves
 * while squad focuses on staying in formation
 * @param ax agent's current x-coord
```

```
 * @param ay agent's current y-coord
 */
public void checkGoalStatus(int ax, int ay){
   if(agentRole == Parameters.SQD_LDR){
     movegoal.checkGoalStatus(ax,ay);
   }
   else{
     formationgoal.checkGoalStatus(ax,ay);
   }
}//end checkGoalStatus


/**
 * getActiveGoal() returns agent's active goal
 * currently squad leader moves
 * while squad focuses on staying in formation
 * @return activeGoal agent's active goal
 */
public int getActiveGoal(){
   int activeGoal;
   if(agentRole ==Parameters.SQD_LDR){
     int moveStatus = movegoal.getStatus();
      activeGoal= Parameters.MOVE_GOAL;
    }
   else{
     int formationStatus = formationgoal.getStatus();
     activeGoal = Parameters.FORMATION_GOAL;
   }
   return activeGoal;
}//end getActiveGoal()

}//end marineGoalManager
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;
import java.util.*;

public class PatrolGoal extends Goal{
  boolean checkPoint1 = false;
  boolean checkPoint2 = false;
  boolean checkPoint3 = false;
  int currentCheckPointX = Parameters.CHECK_PTS[0][0];
  int currentCheckPointY = Parameters.CHECK_PTS[0][1];
  int cpVicinity = 15;

  /**
   * PatrolGoal constructor creates the squad's patrol goal
   * using 3 predetermined checkpoints
   * @param squadVector vector holding Marine agents
   */
  public PatrolGoal(Vector squadVector) {
      Marine tempMarine = (Marine)squadVector.firstElement();
      tempMarine.setNewMoveGoal(currentCheckPointX,
                          currentCheckPointY);
  }//end constructor

  /**
   * checkGoalStatus(Vector squadVector)
   * checks the PatrolGoal status based on current position of
   * squad leader
   * @param squadVector Vector of Marine agents in squad
   */
  public void checkGoalStatus(Vector squadVector){
    Marine sqdldr = (Marine)squadVector.firstElement();
    int distance = (int)Math.sqrt( Math.abs(sqdldr.getX()-currentCheckPointX) *
                             Math.abs(sqdldr.getX()-currentCheckPointX) +
                             Math.abs(sqdldr.getY()-currentCheckPointY) *
                             Math.abs(sqdldr.getY()-currentCheckPointY)
                            );

    if(!checkPoint1){

      if(distance <=cpVicinity){
        checkPoint1 = true;
        currentCheckPointX = Parameters.CHECK_PTS[1][0];
        currentCheckPointY = Parameters.CHECK_PTS[1][1];
        sqdldr.setNewMoveGoal(currentCheckPointX,
                          currentCheckPointY);

      }//end if
```

```
    }
    else{

      if(!checkPoint2){

        if(distance <=cpVicinity){
          checkPoint2 = true;
          currentCheckPointX = Parameters.CHECK_PTS[2][0];
          currentCheckPointY = Parameters.CHECK_PTS[2][1];
          sqdldr.setNewMoveGoal(currentCheckPointX,currentCheckPointY);

        }//end if


      }
      else{
        if(distance <=cpVicinity){
          checkPoint3 = true;
          Parameters.PatrolMission = true;
        }//end if
      }//end if/else
    }//end if/else
  }//end checkGoalStatus


}//end patrolGoal
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:   Copyright (c) 2001
 * Company:     USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 **/

package thesis;

/**
 * class MoveGoal creates the agents' goal to move
 * to a desired location
 * Description:  child class of Goal
 */
public class MoveGoal extends Goal {

  int destX;
  int destY;

  /**
   * MoveGoal constructor
   * @param x goal destination x-coord
   * @param y goal destination y-coord
   */
  public MoveGoal(int x, int y) {
     destX = x;
     destY = y;
  }

  /**
   * getDestX() returns goal x-coord
   * @return destX goal x-coord
   */
   public int getDestX(){
     return destX;
  }


  /**
   * getDestY() returns goal y-coord
   * @return destY goal y-coord
   */
   public int getDestY(){
     return destY;
  }


  /**
   * void checkGoalStatus(int x, int y)
   * determines goal criticality and sets status
   * @param x agent's current x position
   * @param y agent's current y position
   */
```

```
  public void checkGoalStatus(int x, int y){
    int xDist = (int)(Math.abs(x-destX));
    int yDist = (int)(Math.abs(y-destY));
    int newDist = (int)(Math.sqrt(xDist*xDist + yDist*yDist));

    if (newDist > 25){
      setStatus(Parameters.RED);
    }
    else if (newDist <=1){
      setStatus(Parameters.GREEN);
    }
    else{
      setStatus(Parameters.YELLOW);
    }
  }//end checkGoalStatus


}//end MoveGoal
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;


/**
 * class FormationGoal
 * Description:  child class of Goal
 * creates the agent goal to stay in the prescribed
 * squad formation
 */
public class FormationGoal extends Goal {

  private int destX;//goal destination x-coord
  private int destY;//goal destination y-coord
  int heading;      //heading of agent calling on FormationGoal
  int role;         //role of agent who is calling on formationGoal

  /**
   * FormationGoal constructor
   * @param r role of agent creating FormationGoal
   * @param h heading of agent
   * @param baseX x-coordinate of the base of movement
   * @param baseY y-coordinate of the base of movement
   */
   public FormationGoal(int r,int h, int baseX, int baseY) {
     role = r;
     heading = h;
     switch(Parameters.SQD_FORMATION){

       case Parameters.COLUMN:
         setColumn(heading,baseX,baseY);
       break;

       //case Parameters.WEDGE:

       case Parameters.LINE:
         setLine(heading,baseX,baseY);
       break;

     }//end switch
   }//end constructor

   /**
    * setFormation() sets the squad formation based
    * on the position (baseX, baseY) and heading (h)
    * of squad leader agent and the squad formation: Parameters.SQD_FORMATION
    * @param h squad leader heading
```

```
  * @param baseX x-coord of squad leader
  * @param baseY y-coord of squad leader
  */
public void setFormation(int h, int baseX, int baseY){

   switch(Parameters.SQD_FORMATION){
       case Parameters.COLUMN:
          setColumn(h,baseX,baseY);
       break;

       //case Parameters.WEDGE:

       case Parameters.LINE:
          setLine(h,baseX,baseY);
       break;
    }//end switch

}//end setFormation

/**
 * getDestX() returns goal destination x-coord
 * @return destX
 */
  public int getDestX(){
      return destX;
  }//end getDestX

/**
 * getDestY() returns goal destination y-coord
 * @return destY
 */
public int getDestY(){
   return destY;
}//end getDestY

 /**
 * void checkGoalStatus(int x, int y)
 * determines goal criticality and sets status
 * @param x agent's current x position
 * @param y agent's current y position
 */
 public void checkGoalStatus(int x, int y){
    int xDist = (int)(Math.abs(x-destX));
    int yDist = (int)(Math.abs(y-destY));
    int newDist = (int)(Math.sqrt(xDist*xDist + yDist*yDist));

    if (newDist > 15){
       setStatus(Parameters.RED);
    }
    else if (newDist <=1){
       setStatus(Parameters.GREEN);
    }
    else{
       setStatus(Parameters.YELLOW);
    }
 }//end checkGoalStatus
```

```
//public void setWedge(int h, int x, int y)


/**
 * setLine() sets the destination coordinates
 * (destX, destY) of the calling agent based on role in squad
 * to achieve a squad line formation
 * @param h squad leader's heading
 * @param x x-coord of squad leader's position
 * @param y y-coord of squad leader's position
 */
public void setLine(int h, int x, int y){
  int baseX = x;
  int baseY = y;
  switch(role){
     case 0:
        destX = baseX;
        destY = baseY;
        break;

     case 1:
        switch(h){
           case Parameters.NORTH:
              destX = baseX;
              destY = baseY - (Parameters.DISPERSION *
                             1 * Parameters.CELL_WIDTH);
              break;

           case Parameters.SOUTH:
              destX = baseX;
              destY = baseY + (Parameters.DISPERSION *
                          2 * Parameters.CELL_WIDTH);
              break;

           case Parameters.EAST:
              destX = baseX + (Parameters.DISPERSION *
                          2 * Parameters.CELL_WIDTH);
              destY = baseY;
              break;

           case Parameters.WEST:
              destX = baseX - (Parameters.DISPERSION *
                          2 * Parameters.CELL_WIDTH);
              destY = baseY;
              break;

           case Parameters.NORTH_EAST:
              destX = baseX + (Parameters.DISPERSION *2 *
                          Parameters.CELL_WIDTH);
              destY = baseY - (Parameters.DISPERSION *2 *
                          Parameters.CELL_WIDTH);
              break;

           case Parameters.NORTH_WEST:
```

```
            destX = baseX - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;

case 2:
    switch(h){
        case Parameters.NORTH:
            destX = baseX - (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX + (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
            destX = baseX + (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *1 *
                        Parameters.CELL_HEIGHT);
            break;

        case Parameters.WEST:
            destX = baseX - (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *1 *
                        Parameters.CELL_HEIGHT);
            break;

        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
```

```
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX - (Parameters.DISPERSION *4 *
                    Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *4 *
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                    Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;


case 3:
    switch(h){
        case Parameters.NORTH:
            destX = baseX;
            destY = baseY - (Parameters.DISPERSION *4 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX;
            destY = baseY + (Parameters.DISPERSION *4 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
             destX = baseX + (Parameters.DISPERSION*4 *
                     Parameters.CELL_WIDTH);
             destY = baseY;
            break;

        case Parameters.WEST:
             destX = baseX - (Parameters.DISPERSION *4 *
                     Parameters.CELL_WIDTH);
             destY = baseY;
            break;

        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *4 *
                    Parameters.CELL_WIDTH);
```

```
            destY = baseY - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;
        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;

case 4:
    switch(h){
        case Parameters.NORTH:
            destX = baseX + (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX - (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
             destX = baseX + (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
             destY = baseY + (Parameters.DISPERSION *1 *
                        Parameters.CELL_HEIGHT);
            break;

        case Parameters.WEST:
             destX = baseX - (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
             destY = baseY - (Parameters.DISPERSION *1 *
                        Parameters.CELL_HEIGHT);
            break;
```

```
        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;


  case 5:
      switch(h){
        case Parameters.NORTH:
            destX = baseX - (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX + (Parameters.DISPERSION *4 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
              destX = baseX + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                        Parameters.CELL_HEIGHT);
            break;

        case Parameters.WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
```

```
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *4 *
                Parameters.CELL_HEIGHT);
        break;

    case Parameters.NORTH_EAST:
        destX = baseX - (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.NORTH_WEST:
        destX = baseX - (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH_EAST:
        destX = baseX + (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH_WEST:
        destX = baseX + (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        break;
  }//end switch
  break;

case 6:
  switch(h){
    case Parameters.NORTH:
        destX = baseX - (Parameters.DISPERSION *5 *
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *3 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH:
        destX = baseX + (Parameters.DISPERSION *5 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *3 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.EAST:
        destX = baseX + (Parameters.DISPERSION *3 *
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *5 *
                Parameters.CELL_HEIGHT);
```

```
                break;

            case Parameters.WEST:
                 destX = baseX - (Parameters.DISPERSION *3 *
                            Parameters.CELL_WIDTH);
                 destY = baseY + (Parameters.DISPERSION *5 *
                            Parameters.CELL_HEIGHT);
                break;

            case Parameters.NORTH_EAST:
                destX = baseX - (Parameters.DISPERSION *2 *
                           Parameters.CELL_WIDTH);
                destY = baseY - (Parameters.DISPERSION *8 *
                           Parameters.CELL_WIDTH);
                break;

            case Parameters.NORTH_WEST:
               destX = baseX - (Parameters.DISPERSION *8 *
                          Parameters.CELL_WIDTH);
               destY = baseY + (Parameters.DISPERSION *2 *
                          Parameters.CELL_WIDTH);
                break;

            case Parameters.SOUTH_EAST:
               destX = baseX + (Parameters.DISPERSION *8 *
                          Parameters.CELL_WIDTH);
               destY = baseY - (Parameters.DISPERSION *2 *
                          Parameters.CELL_WIDTH);
                break;

            case Parameters.SOUTH_WEST:
               destX = baseX + (Parameters.DISPERSION *2 *
                          Parameters.CELL_WIDTH);
                destY = baseY + (Parameters.DISPERSION *8 *
                           Parameters.CELL_WIDTH);
                break;
        }//end switch
        break;

    case 7:
        switch(h){
            case Parameters.NORTH:
               destX = baseX - (Parameters.DISPERSION *4 *
                          Parameters.CELL_WIDTH);
               destY = baseY - (Parameters.DISPERSION *4 *
                          Parameters.CELL_WIDTH);
                break;

            case Parameters.SOUTH:
               destX = baseX + (Parameters.DISPERSION *4 *
                          Parameters.CELL_WIDTH);
               destY = baseY + (Parameters.DISPERSION *4 *
                         Parameters.CELL_WIDTH);
                break;

            case Parameters.EAST:
```

```
             destX = baseX + (Parameters.DISPERSION *4 *
                             Parameters.CELL_WIDTH);
             destY = baseY - (Parameters.DISPERSION *4 *
                             Parameters.CELL_HEIGHT);
            break;

        case Parameters.WEST:
             destX = baseX - (Parameters.DISPERSION *4 *
                             Parameters.CELL_WIDTH);
             destY = baseY + (Parameters.DISPERSION *4 *
                             Parameters.CELL_HEIGHT);
            break;

        case Parameters.NORTH_EAST:
            destX = baseX;
            destY = baseY - (Parameters.DISPERSION *8 *
                            Parameters.CELL_HEIGHT);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX - (Parameters.DISPERSION *8 *
                            Parameters.CELL_HEIGHT);
            destY = baseY;
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *8 *
                            Parameters.CELL_HEIGHT);
            destY = baseY;
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX;
            destY = baseY + (Parameters.DISPERSION *8 *
                           Parameters.CELL_HEIGHT);
            break;
    }//end switch
    break;

case 8:
    switch(h){
        case Parameters.NORTH:
            destX = baseX - (Parameters.DISPERSION *3 *
                             Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *3 *
                              Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX + (Parameters.DISPERSION *3 *
                            Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *3 *
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
```

```
            destX = baseX + (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *3 *
                    Parameters.CELL_HEIGHT);
           break;

        case Parameters.WEST:
            destX = baseX - (Parameters.DISPERSION *3 *
                     Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *3 *
                    Parameters.CELL_HEIGHT);
           break;

        case Parameters.NORTH_EAST:
            destX = baseX;
            destY = baseY - (Parameters.DISPERSION *6 *
                    Parameters.CELL_HEIGHT);
           break;

        case Parameters.NORTH_WEST:
           destX = baseX - (Parameters.DISPERSION *6 *
                   Parameters.CELL_HEIGHT);
           destY = baseY;
           break;

        case Parameters.SOUTH_EAST:
           destX = baseX + (Parameters.DISPERSION *6 *
                   Parameters.CELL_HEIGHT);
           destY = baseY;
           break;

        case Parameters.SOUTH_WEST:
           destX = baseX;
           destY = baseY + (Parameters.DISPERSION *6 *
                    Parameters.CELL_HEIGHT);
           break;
    }//end switch
    break;

case 9:
   switch(h){
        case Parameters.NORTH:
          destX = baseX +(Parameters.DISPERSION *4 *
                  Parameters.CELL_WIDTH);
          destY = baseY - (Parameters.DISPERSION *2 *
                   Parameters.CELL_WIDTH);
           break;

        case Parameters.SOUTH:
          destX = baseX -(Parameters.DISPERSION *4 *
                  Parameters.CELL_WIDTH);
          destY = baseY + (Parameters.DISPERSION *2 *
                   Parameters.CELL_WIDTH);
           break;

        case Parameters.EAST:
```

```
            destX = baseX + (Parameters.DISPERSION *2 *
                      Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                       Parameters.CELL_HEIGHT);
           break;

        case Parameters.WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
                      Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                      Parameters.CELL_HEIGHT);
           break;

        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                      Parameters.CELL_HEIGHT);
           break;

        case Parameters.NORTH_WEST:
            destX = baseX + (Parameters.DISPERSION *2 *
                      Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *6 *
                      Parameters.CELL_HEIGHT);
           break;

        case Parameters.SOUTH_EAST:
            destX = baseX - (Parameters.DISPERSION *2 *
                      Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *6 *
                      Parameters.CELL_HEIGHT);
           break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                      Parameters.CELL_HEIGHT);
           break;
    }//end switch
    break;

case 10:
   switch(h){
        case Parameters.NORTH:
           destX = baseX + (Parameters.DISPERSION *3 *
                     Parameters.CELL_WIDTH);
           destY = baseY - (Parameters.DISPERSION *3 *
                      Parameters.CELL_WIDTH);
           break;

        case Parameters.SOUTH:
           destX = baseX - (Parameters.DISPERSION *3 *
                      Parameters.CELL_WIDTH);
           destY = baseY + (Parameters.DISPERSION *3 *
```

```
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
             destX = baseX + (Parameters.DISPERSION *3 *
                       Parameters.CELL_WIDTH);
             destY = baseY + (Parameters.DISPERSION *3 *
                       Parameters.CELL_HEIGHT);
            break;

        case Parameters.WEST:
             destX = baseX - (Parameters.DISPERSION *3 *
                       Parameters.CELL_WIDTH);
             destY = baseY - (Parameters.DISPERSION *3 *
                       Parameters.CELL_HEIGHT);
            break;

        case Parameters.NORTH_EAST:
             destX = baseX + (Parameters.DISPERSION *6 *
                       Parameters.CELL_WIDTH);
             destY = baseY;
            break;

        case Parameters.NORTH_WEST:
            destX = baseX;
            destY = baseY - (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX;
            destY = baseY + (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
            break;


        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
            destY = baseY;
            break;

    }//end switch
    break;

case 11:
    switch(h){
        case Parameters.NORTH:
            destX = baseX + (Parameters.DISPERSION *4 *
                       Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                      Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
           destX = baseX - (Parameters.DISPERSION *4 *
```

```
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *4 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.EAST:
        destX = baseX + (Parameters.DISPERSION *4 *
                 Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *4 *
                 Parameters.CELL_HEIGHT);
        break;

      case Parameters.WEST:
        destX = baseX - (Parameters.DISPERSION *4 *
                Parameters.CELL_WIDTH);
        destY = baseY -(Parameters.DISPERSION *4 *
                Parameters.CELL_HEIGHT);
        break;

      case Parameters.NORTH_EAST:
        destX = baseX + (Parameters.DISPERSION *8 *
                 Parameters.CELL_WIDTH);
        destY = baseY;
        break;

      case Parameters.NORTH_WEST:
        destX = baseX;
        destY = baseY - (Parameters.DISPERSION *8 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.SOUTH_EAST:
        destX = baseX;
        destY = baseY + (Parameters.DISPERSION *8 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.SOUTH_WEST:
        destX = baseX - (Parameters.DISPERSION *8 *
                Parameters.CELL_WIDTH);
        destY = baseY;
        break;
    }//end switch
    break;

  case 12:
    switch(h){
      case Parameters.NORTH:
        destX = baseX + (Parameters.DISPERSION *5 *
                 Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *3 *
                  Parameters.CELL_WIDTH);
        break;

      case Parameters.SOUTH:
        destX = baseX - (Parameters.DISPERSION *5 *
```

```
                    Parameters.CELL_WIDTH);
          destY = baseY + (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
          break;

        case Parameters.EAST:
          destX = baseX + (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
          destY = baseY + (Parameters.DISPERSION *5 *
                    Parameters.CELL_HEIGHT);
          break;

        case Parameters.WEST:
          destX = baseX - (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
          destY = baseY - (Parameters.DISPERSION *5 *
                    Parameters.CELL_HEIGHT);
          break;

        case Parameters.NORTH_EAST:
          destX = baseX + (Parameters.DISPERSION *8 *
                    Parameters.CELL_WIDTH);
          destY = baseY + (Parameters.DISPERSION *2 *
                    Parameters.CELL_HEIGHT);
          break;

        case Parameters.NORTH_WEST:
          destX = baseX + (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
          destY = baseY - (Parameters.DISPERSION *8 *
                    Parameters.CELL_HEIGHT);
          break;

        case Parameters.SOUTH_EAST:
          destX = baseX - (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
          destY = baseY + (Parameters.DISPERSION *8 *
                    Parameters.CELL_HEIGHT);
          break;

        case Parameters.SOUTH_WEST:
          destX = baseX - (Parameters.DISPERSION *8 *
                    Parameters.CELL_WIDTH);
          destY = baseY - (Parameters.DISPERSION *2 *
                    Parameters.CELL_HEIGHT);
          break;

      }//end switch
      break;

  }//end switch
}//end setLine()


/**
 * setColumn() sets the destination coordinates
```

```
 * (destX, destY) of the calling agent based on role in squad
 * to achieve a squad column formation
 * @param h squad leader's heading
 * @param x x-coord of squad leader's position
 * @param y y-coord of squad leader's position
 */

public void setColumn(int h, int x, int y){
    int baseX = x;
     int baseY = y;

     switch(role){

        case 0:
           destX = baseX;
           destY = baseY;
           break;

        case 1:
           switch(h){
              case Parameters.NORTH:
                  destX = baseX +(Parameters.DISPERSION *
                                 Parameters.CELL_WIDTH);
                  destY = baseY -(Parameters.DISPERSION *2 *
                                 Parameters.CELL_WIDTH);
                  break;

              case Parameters.SOUTH:
                  destX = baseX-(Parameters.DISPERSION *
                               Parameters.CELL_WIDTH);
                  destY = baseY +(Parameters.DISPERSION *2 *
                                 Parameters.CELL_WIDTH);
                  break;

              case Parameters.EAST:
                  destX = baseX + (Parameters.DISPERSION *2 *
                                 Parameters.CELL_WIDTH);
                  destY = baseY+(Parameters.DISPERSION *1*
                                Parameters.CELL_WIDTH);
                  break;

              case Parameters.WEST:
                  destX = baseX - (Parameters.DISPERSION *2 *
                                 Parameters.CELL_WIDTH);
                  destY = baseY-(Parameters.DISPERSION *1*
                                 Parameters.CELL_WIDTH);
                  break;

              case Parameters.NORTH_EAST:
                  destX = baseX + (Parameters.DISPERSION *2 *
                                  Parameters.CELL_WIDTH);
                  destY = baseY - (Parameters.DISPERSION *1 *
                                   Parameters.CELL_WIDTH);
                  break;

              case Parameters.NORTH_WEST:
```

```
            destX = baseX - (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
                        Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;

  case 2:
    switch(h){
        case Parameters.NORTH:
            destX = baseX + (Parameters.DISPERSION *1*
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *5 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX-(Parameters.DISPERSION *1*
                       Parameters.CELL_WIDTH);
            destY = baseY +(Parameters.DISPERSION *5 *
                       Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
            destX = baseX + (Parameters.DISPERSION *5 *
                        Parameters.CELL_WIDTH);
            destY = baseY+  (Parameters.DISPERSION *1*
                       Parameters.CELL_WIDTH);
            break;

        case Parameters.WEST:
             destX = baseX - (Parameters.DISPERSION *5*
                        Parameters.CELL_WIDTH);
            destY = baseY-(Parameters.DISPERSION *1*
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *3 *
                        Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
```

```
                Parameters.CELL_WIDTH);
        break;

    case Parameters.NORTH_WEST:
        destX = baseX - (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *3 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH_EAST:
        destX = baseX + (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *3 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH_WEST:
        destX = baseX - (Parameters.DISPERSION *2 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *3 *
                Parameters.CELL_WIDTH);
        break;
  }//end switch
  break;


case 3:
  switch(h){
    case Parameters.NORTH:
        destX = baseX - (Parameters.DISPERSION *1*
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH:
        destX = baseX + (Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.EAST:
        destX = baseX + (Parameters.DISPERSION *6*
                Parameters.CELL_WIDTH);
        destY = baseY -(Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        break;

    case Parameters.WEST:
        destX = baseX - (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        destY = baseY +(Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        break;
```

```
        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *3 *
                            Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX - (Parameters.DISPERSION *4 *
                            Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *3 *
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *4 *
                            Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *3*
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *3 *
                            Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                            Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;

case 4:
    switch(h){
        case Parameters.NORTH:
            destX = baseX - (Parameters.DISPERSION *1*
                            Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2*
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX + (Parameters.DISPERSION *1 *
                            Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
             destX = baseX + (Parameters.DISPERSION *4 *
                            Parameters.CELL_WIDTH);
             destY = baseY -(Parameters.DISPERSION *1 *
                            Parameters.CELL_WIDTH);
            break;

        case Parameters.WEST:
             destX = baseX - (Parameters.DISPERSION *2 *
```

```
                    Parameters.CELL_WIDTH);
            destY = baseY +(Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_EAST:
            destX = baseX + (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX - (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX + (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX - (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;


case 5:
      switch(h){
        case Parameters.NORTH:
            destX = baseX + (Parameters.DISPERSION *1*
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX-(Parameters.DISPERSION *1*
                    Parameters.CELL_WIDTH);
            destY = baseY -(Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
            destX = baseX - (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
            destY = baseY+(Parameters.DISPERSION *1*
```

```
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.WEST:
             destX = baseX+ (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
            destY = baseY-(Parameters.DISPERSION *1*
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_EAST:
            destX = baseX - (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *2*
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX + (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *1 *
                     Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX - (Parameters.DISPERSION *2 *
                     Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
            destX = baseX + (Parameters.DISPERSION *1 *
                     Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *2 *
                    Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;

case 6:
    switch(h){
        case Parameters.NORTH:
            destX = baseX + (Parameters.DISPERSION *1*
                     Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *6 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
            destX = baseX-(Parameters.DISPERSION *1*
                     Parameters.CELL_WIDTH);
            destY = baseY -(Parameters.DISPERSION *6 *
                    Parameters.CELL_WIDTH);
            break;
```

```
        case Parameters.EAST:
            destX = baseX - (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
            destY = baseY+(Parameters.DISPERSION *1*
                     Parameters.CELL_WIDTH);
            break;

        case Parameters.WEST:
            destX = baseX + (Parameters.DISPERSION *6 *
                     Parameters.CELL_WIDTH);
            destY = baseY-(Parameters.DISPERSION *1*
                     Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_EAST:
            destX = baseX - (Parameters.DISPERSION *3 *
                       Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *4 *
                      Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_WEST:
           destX = baseX + (Parameters.DISPERSION *4 *
                     Parameters.CELL_WIDTH);
           destY = baseY + (Parameters.DISPERSION *3 *
                      Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_EAST:
           destX = baseX - (Parameters.DISPERSION *4 *
                     Parameters.CELL_WIDTH);
           destY = baseY - (Parameters.DISPERSION *3 *
                    Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH_WEST:
           destX = baseX + (Parameters.DISPERSION *3 *
                      Parameters.CELL_WIDTH);
            destY = baseY - (Parameters.DISPERSION *4 *
                     Parameters.CELL_WIDTH);
            break;
    }//end switch
    break;

  case 7:
    switch(h){
        case Parameters.NORTH:
           destX = baseX - (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
           destY = baseY + (Parameters.DISPERSION *2 *
                   Parameters.CELL_WIDTH);
            break;

        case Parameters.SOUTH:
           destX = baseX + (Parameters.DISPERSION *1 *
                    Parameters.CELL_WIDTH);
```

```
            destY = baseY - (Parameters.DISPERSION *2 *
                             Parameters.CELL_WIDTH);
             break;

        case Parameters.EAST:
              destX = baseX - (Parameters.DISPERSION *2 *
                             Parameters.CELL_WIDTH);
              destY = baseY -(Parameters.DISPERSION *1 *
                            Parameters.CELL_WIDTH);
             break;

        case Parameters.WEST:
              destX = baseX + (Parameters.DISPERSION *2 *
                             Parameters.CELL_WIDTH);
              destY = baseY +(Parameters.DISPERSION *1 *
                           Parameters.CELL_WIDTH);
             break;

        case Parameters.NORTH_EAST:
             destX = baseX -(Parameters.DISPERSION *1 *
                            Parameters.CELL_HEIGHT);
             destY = baseY + (Parameters.DISPERSION *1 *
                              Parameters.CELL_HEIGHT);
             break;

        case Parameters.NORTH_WEST:
             destX = baseX + (Parameters.DISPERSION *1 *
                             Parameters.CELL_HEIGHT);
             destY = baseY+(Parameters.DISPERSION *1 *
                            Parameters.CELL_HEIGHT);
             break;

        case Parameters.SOUTH_EAST:
             destX = baseX - (Parameters.DISPERSION *1*
                            Parameters.CELL_HEIGHT);
             destY = baseY-(Parameters.DISPERSION *1 *
                           Parameters.CELL_HEIGHT);
             break;

        case Parameters.SOUTH_WEST:
             destX = baseX+ (Parameters.DISPERSION *1 *
                          Parameters.CELL_HEIGHT);
             destY = baseY - (Parameters.DISPERSION *1*
                             Parameters.CELL_HEIGHT);
             break;
    }//end switch
    break;

case 8:
    switch(h){
        case Parameters.NORTH:
              destX = baseX - (Parameters.DISPERSION *1 *
                              Parameters.CELL_WIDTH);
            destY = baseY + (Parameters.DISPERSION *5 *
                             Parameters.CELL_WIDTH);
             break;
```

```
        case Parameters.SOUTH:
           destX = baseX + (Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
           destY = baseY - (Parameters.DISPERSION *5 *
                         Parameters.CELL_WIDTH);
            break;

        case Parameters.EAST:
             destX = baseX - (Parameters.DISPERSION *5 *
                         Parameters.CELL_WIDTH);
             destY = baseY -(Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.WEST:
             destX = baseX + (Parameters.DISPERSION *5 *
                         Parameters.CELL_WIDTH);
             destY = baseY +(Parameters.DISPERSION *1 *
                        Parameters.CELL_WIDTH);
            break;

        case Parameters.NORTH_EAST:
             destX = baseX- (Parameters.DISPERSION *3 *
                         Parameters.CELL_HEIGHT);
             destY = baseY + (Parameters.DISPERSION *2 *
                          Parameters.CELL_HEIGHT);
            break;

        case Parameters.NORTH_WEST:
            destX = baseX + (Parameters.DISPERSION *2 *
                        Parameters.CELL_HEIGHT);
            destY = baseY+(Parameters.DISPERSION *3 *
                         Parameters.CELL_HEIGHT);
            break;

        case Parameters.SOUTH_EAST:
            destX = baseX -(Parameters.DISPERSION *2*
                         Parameters.CELL_HEIGHT);
            destY = baseY-(Parameters.DISPERSION *3 *
                         Parameters.CELL_HEIGHT);
            break;


        case Parameters.SOUTH_WEST:
            destX = baseX+ (Parameters.DISPERSION *3 *
                        Parameters.CELL_HEIGHT);
            destY = baseY - (Parameters.DISPERSION *2*
                       Parameters.CELL_HEIGHT);
            break;
      }//end switch
      break;

  case 9:
     switch(h){
        case Parameters.NORTH:
```

```
        destX = baseX + (Parameters.DISPERSION *1*
                       Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *9 *
                      Parameters.CELL_WIDTH);
        break;

    case Parameters.SOUTH:
        destX = baseX-(Parameters.DISPERSION *1*
                    Parameters.CELL_WIDTH);
        destY = baseY -(Parameters.DISPERSION *9 *
                     Parameters.CELL_WIDTH);
        break;

    case Parameters.EAST:
        destX = baseX - (Parameters.DISPERSION *9 *
                       Parameters.CELL_WIDTH);
        destY = baseY+(Parameters.DISPERSION *1*
                      Parameters.CELL_WIDTH);
        break;

    case Parameters.WEST:
        destX = baseX + (Parameters.DISPERSION *9 *
                      Parameters.CELL_WIDTH);
        destY = baseY-(Parameters.DISPERSION *1*
                       Parameters.CELL_WIDTH);
        break;

    case Parameters.NORTH_EAST:
        destX = baseX - (Parameters.DISPERSION *5 *
                      Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *6 *
                      Parameters.CELL_HEIGHT);
        break;

    case Parameters.NORTH_WEST:
        destX = baseX + (Parameters.DISPERSION *6 *
                      Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *5 *
                      Parameters.CELL_HEIGHT);
        break;

    case Parameters.SOUTH_EAST:
        destX = baseX - (Parameters.DISPERSION *6 *
                       Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *5 *
                      Parameters.CELL_HEIGHT);
        break;

    case Parameters.SOUTH_WEST:
        destX = baseX + (Parameters.DISPERSION *5 *
                      Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *6 *
                      Parameters.CELL_HEIGHT);
        break;
}//end switch
break;
```

```
case 10:
  switch(h){
     case Parameters.NORTH:
        destX = baseX + (Parameters.DISPERSION *1*
                 Parameters.CELL_WIDTH);
         destY = baseY + (Parameters.DISPERSION *12 *
                 Parameters.CELL_WIDTH);
         break;

     case Parameters.SOUTH:
         destX = baseX-(Parameters.DISPERSION *1*
                  Parameters.CELL_WIDTH);
         destY = baseY -(Parameters.DISPERSION *12 *
                  Parameters.CELL_WIDTH);
         break;

     case Parameters.EAST:
         destX = baseX - (Parameters.DISPERSION *12 *
                   Parameters.CELL_WIDTH);
         destY = baseY+(Parameters.DISPERSION *1*
                     Parameters.CELL_WIDTH);
         break;

     case Parameters.WEST:
          destX = baseX + (Parameters.DISPERSION *12 *
                  Parameters.CELL_WIDTH);
         destY = baseY-(Parameters.DISPERSION *1*
                  Parameters.CELL_WIDTH);
         break;

     case Parameters.NORTH_EAST:
          destX = baseX - (Parameters.DISPERSION *7 *
                   Parameters.CELL_WIDTH);
          destY = baseY+ (Parameters.DISPERSION *8 *
                   Parameters.CELL_WIDTH);
         break;

     case Parameters.NORTH_WEST:
         destX = baseX+(Parameters.DISPERSION *8 *
                 Parameters.CELL_HEIGHT);
         destY = baseY + (Parameters.DISPERSION *7*
                  Parameters.CELL_WIDTH);
         break;

     case Parameters.SOUTH_EAST:
         destX = baseX-(Parameters.DISPERSION *8 *
                  Parameters.CELL_HEIGHT);
         destY = baseY - (Parameters.DISPERSION *7 *
                  Parameters.CELL_WIDTH);
         break;


     case Parameters.SOUTH_WEST:
         destX = baseX + (Parameters.DISPERSION *7 *
```

```
                 Parameters.CELL_WIDTH);
          destY = baseY- (Parameters.DISPERSION *8 *
                 Parameters.CELL_WIDTH);
          break;

      }//end switch
      break;

   case 11:
      switch(h){
        case Parameters.NORTH:
          destX = baseX - (Parameters.DISPERSION *1 *
                 Parameters.CELL_WIDTH);
          destY = baseY + (Parameters.DISPERSION *8 *
                 Parameters.CELL_WIDTH);
          break;

        case Parameters.SOUTH:
          destX = baseX + (Parameters.DISPERSION *1 *
                 Parameters.CELL_WIDTH);
          destY = baseY - (Parameters.DISPERSION *8 *
                 Parameters.CELL_WIDTH);
          break;

        case Parameters.EAST:
          destX = baseX - (Parameters.DISPERSION *8*
                 Parameters.CELL_WIDTH);
          destY = baseY -(Parameters.DISPERSION *1 *
                 Parameters.CELL_WIDTH);
          break;

        case Parameters.WEST:
          destX = baseX + (Parameters.DISPERSION *8 *
                 Parameters.CELL_WIDTH);
          destY = baseY +(Parameters.DISPERSION *1 *
                 Parameters.CELL_WIDTH);
          break;

        case Parameters.NORTH_EAST:
          destX = baseX - (Parameters.DISPERSION *5 *
                 Parameters.CELL_WIDTH);
          destY = baseY+  (Parameters.DISPERSION *4 *
                 Parameters.CELL_WIDTH);
          break;

        case Parameters.NORTH_WEST:
          destX = baseX+(Parameters.DISPERSION *4 *
                 Parameters.CELL_HEIGHT);
          destY = baseY +(Parameters.DISPERSION *5 *
                 Parameters.CELL_WIDTH);
          break;

        case Parameters.SOUTH_EAST:
          destX = baseX-(Parameters.DISPERSION *4 *
                 Parameters.CELL_HEIGHT);
          destY = baseY - (Parameters.DISPERSION *5 *
```

```
                Parameters.CELL_WIDTH);
        break;

      case Parameters.SOUTH_WEST:
        destX = baseX+ (Parameters.DISPERSION *5 *
                Parameters.CELL_HEIGHT);
        destY = baseY - (Parameters.DISPERSION *4*
                Parameters.CELL_HEIGHT);
        break;
    }//end switch
    break;

case 12:
    switch(h){
      case Parameters.NORTH:
        destX = baseX - (Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *11 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.SOUTH:
        destX = baseX + (Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        destY = baseY - (Parameters.DISPERSION *11 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.EAST:
        destX = baseX - (Parameters.DISPERSION *11*
                Parameters.CELL_WIDTH);
        destY = baseY -(Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.WEST:
        destX = baseX + (Parameters.DISPERSION *11 *
                Parameters.CELL_WIDTH);
        destY = baseY +(Parameters.DISPERSION *1 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.NORTH_EAST:
        destX = baseX - (Parameters.DISPERSION *7 *
                Parameters.CELL_WIDTH);
        destY = baseY+  (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        break;

      case Parameters.NORTH_WEST:
        destX = baseX + (Parameters.DISPERSION *6 *
                Parameters.CELL_WIDTH);
        destY = baseY + (Parameters.DISPERSION *7 *
                Parameters.CELL_HEIGHT);
        break;
```

```
            case Parameters.SOUTH_EAST:
                destX = baseX - (Parameters.DISPERSION *6 *
                                Parameters.CELL_WIDTH);
                destY = baseY - (Parameters.DISPERSION *7 *
                               Parameters.CELL_HEIGHT);
                break;

            case Parameters.SOUTH_WEST:
                destX = baseX+ (Parameters.DISPERSION *7 *
                              Parameters.CELL_HEIGHT);
                destY = baseY - (Parameters.DISPERSION *6*
                              Parameters.CELL_HEIGHT);
                break;
          }//end switch
          break;
      }//end switch
  }//end setColumn()

}//end FormationGoal
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

/**
 * class Goal
 * Description:  class Goal
 * parent class for agent goal objects
 */

public class Goal {

  int status;  //goal status
  int type;    //type of goal

  //Goal constructor
  public Goal() {
  }

  /**
   * void setStatus(int x)
   * @param x status of goal
   */
  public void setStatus(int x){
     status = x;
  }//end setStatus

  /**
   * void setGoalType(int t)
   * @param t type of goal
   */
  public void setGoalType(int t){
     type = t;
  }//end setGoalType


  /**
   * int getStatus()
   * @return status of goal
   */
   public int getStatus(){
     return status;
   }//end getStatus

   /**
   * int getGoalType()
   * @return type goal type
   */
```

```
  public int getGoalType(){
    return type;
  }//end getGoalType


}//end Goal
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.util.*;
import java.awt.*;

/**
 * class PathSearchGraph creates a searchgraph of PathNodes
 * for agent path-planning
 */

public class PathSearchGraph extends Hashtable {
  String name;

  public PathSearchGraph(String n) {
    name = n;
  }

  //reset each PathNode in graph
  void reset() {
    Enumeration e = this.elements();
    while(e.hasMoreElements() ){
      PathNode nextNode = (PathNode)e.nextElement();
      nextNode.reset();
    }//end while
  }//end reset


  //add node to hashtable using nodeID as key
  void put(PathNode node){
    put(node.getNodeID(), node);
  }//end put


  /**
   * setLinks() establishes links between
   * nodes that have a clear line of sight and
   * are within los distance
   */
  public void setLinks(){
    Enumeration e = this.elements();
    while(e.hasMoreElements()){
      PathNode nextNode = (PathNode)e.nextElement();

      Enumeration e2 = this.elements();
      while(e2.hasMoreElements()){
```

```
        PathNode linkNode = (PathNode)e2.nextElement();
        if( linkNode.getNodeID() != nextNode.getNodeID() &&
          lineOfSight(nextNode.getX(),nextNode.getY(),
             linkNode.getX(), linkNode.getY()) &&
          distance(nextNode.getX(),nextNode.getY(),
             linkNode.getX(), linkNode.getY()) <  Parameters.LOS_DIST){
           nextNode.addLink(linkNode);
        }
      }//end while
   }//end while
}//end setLinks



/**
 * setDistanceCost(PathNode goalNode) calculates
 *  the distance cost to the goalNode from each Node in
 *  the search graph
 */
public void setDistanceCost(PathNode goalNode){

   Enumeration e = this.elements();
   while(e.hasMoreElements()){
     PathNode nextNode = (PathNode)e.nextElement();
     nextNode.setDistanceCost( distance(nextNode.getX(),
                                 nextNode.getY(),
                                 goalNode.getX(),
                                 goalNode.getY() ) );
   }//
}//end setDistanceCost


/**
 * int distance(cx,cy,gx,gy) returns the distance
 * between the points(cx,cy) and (gx,gy)
 * @return distance distance between two points
 */
public int distance(int cx, int cy, int gx, int gy){
   int distance=0;
   double delX = (cx-gx);
   double delY = (cy-gy);
   distance = (int)Math.sqrt(delX*delX + delY*delY);
   return distance;
}//end distance


/**
 * boolean lineOfSight(cx,cy,gx,gy) determines
 * if there is a clear line of sight between the two points
 * (cx,cy) and (gx,gy)
 * @return isLOS true if Line of Sight is clear
 */
public boolean lineOfSight(int cx, int cy, int gx, int gy){
  boolean isLOS = true;
  double delX = (cx-gx);
  double delY = (cy-gy);
```

```
  double iy;
  if(delX == 0){
     for( double ix = 5; ix <= Math.abs(delY); ix+=1){
     iy =  ix;
     if(delY<0){
       if(Parameters.SIM_ENV[(int)(cy +iy)/5][(int)cx/5]>10){
        isLOS = false;
       }//end if
     }
     else{
       if(Parameters.SIM_ENV[(int)(cy-iy)/5][(int)cx/5] >10){
         isLOS = false;
       }//end if
     }//end if/else
   }//end for
  }
  else{
    double slope = delY/delX;
    for( double ix = 5; ix <= Math.abs(delX); ix+=1){
     iy =  (slope*ix);
     if(delX<0){
       if(Parameters.SIM_ENV[(int)(cy+iy)/5][(int)(cx+ix)/5] >10){
         isLOS = false;
       }
     }
     else{
       if(Parameters.SIM_ENV[(int)(cy-iy)/5][(int)(cx-ix)/5] >10){
         isLOS = false;
       }
     }//end if/else
   }//end for
 }//endif/else

 return isLOS;
}//lineOfSight

/**
 * Vector breadthFirstSearch(startNode, endNode)
 * conducts a BFS from startNode to endNode
 * and returns a vector containing the nodes that lead from
 * the startNode to endNode
 * @param startNode agent's starting node
 * @param endNode agent's desired location
 * @return pathToGoal Vector containing the list of nodes leading to endNode
 */
public Vector breadthFirstSearch(PathNode startNode, PathNode endNode){
   Vector temp = new Vector();
   Vector pathToGoal = new Vector();
   Vector queue = new Vector();
   queue.addElement(startNode);
   startNode.setTested(true);

   while(queue.size()>0){
     PathNode testNode = (PathNode)queue.firstElement();
     queue.removeElementAt(0);
     temp.addElement(testNode);
```

```
    if(testNode.getNodeID() == endNode.getNodeID()){
      pathToGoal.addElement(testNode);
      PathNode backNode = (PathNode)testNode.getBackTrace();
      pathToGoal.addElement(backNode);
      while(backNode.getBackTrace()!= null){
        backNode = (PathNode)backNode.getBackTrace();
        pathToGoal.addElement(backNode);
      }//

      return pathToGoal;
    }

    if(!testNode.expanded){
      testNode.expand(queue,PathNode.BACK);
    }

  }//end while

  return null;
}//end bfs


/**
 * Vector depthFirstSearch(startNode, endNode)
 * conducts a DFS from startNode to endNode
 * and returns a vector containing the nodes that lead from
 * the startNode to endNode
 * @param startNode agent's starting node
 * @param endNode agent's desired location
 * @return pathToGoal Vector containing the list of nodes leading to endNode
 */
public Vector depthFirstSearch(PathNode startNode, PathNode endNode){

  Vector temp = new Vector();
  Vector pathToGoal = new Vector();
  Vector queue = new Vector();
  queue.addElement(startNode);
  startNode.setTested(true);

  while(queue.size()>0){
    PathNode testNode = (PathNode)queue.firstElement();
    queue.removeElementAt(0);
    temp.addElement(testNode);

    if(testNode.getNodeID() == endNode.getNodeID()){
      pathToGoal.addElement(testNode);
      PathNode backNode = (PathNode)testNode.getBackTrace();
      pathToGoal.addElement(backNode);
      while(backNode.getBackTrace()!= null){
        backNode = (PathNode)backNode.getBackTrace();
        pathToGoal.addElement(backNode);
      }//

      return pathToGoal;
    }
```

```
    if(!testNode.expanded){
      testNode.expand(queue,PathNode.FRONT);
    }

  }//end while

  return null;

}//end dfs


/**
 * Vector bestFirstSearch(startNode, endNode)
 * conducts a BestFS (Shortest Distance) from startNode to endNode
 * and returns a vector containing the nodes that lead from
 * the startNode to endNode
 * @param startNode agent's starting node
 * @param endNode agent's desired location
 * @return pathToGoal Vector containing the list of nodes leading to endNode
 */
public Vector bestFirstSearch(PathNode startNode, PathNode endNode){

  Vector temp = new Vector();
  Vector pathToGoal = new Vector();
  Vector queue = new Vector();

  setDistanceCost(endNode);

  queue.addElement(startNode);
  startNode.setTested(true);

  while(queue.size()>0){
    PathNode testNode = (PathNode)queue.firstElement();
    queue.removeElementAt(0);
    temp.addElement(testNode);

    if(testNode.getNodeID() == endNode.getNodeID()){
      pathToGoal.addElement(testNode);
      PathNode backNode = (PathNode)testNode.getBackTrace();
      pathToGoal.addElement(backNode);
      while(backNode.getBackTrace()!= null){
        backNode = (PathNode)backNode.getBackTrace();
        pathToGoal.addElement(backNode);
      }//

      return pathToGoal;
    }

    if(!testNode.expanded){
      testNode.expand(queue,PathNode.INSERT_DIST);
    }

  }//end while

  return null;
```

```
}//end bestfs


/**
 * Vector coverFirstSearch(startNode, endNode)
 * conducts a BestFS (highest cover value) from startNode to endNode
 * and returns a vector containing the nodes that lead from
 * the startNode to endNode
 * @param startNode agent's starting node
 * @param endNode agent's desired location
 * @return pathToGoal Vector containing the list of nodes leading to endNode
 */

 public Vector coverFirstSearch(PathNode startNode, PathNode endNode){

   Vector temp = new Vector();
   Vector pathToGoal = new Vector();
   Vector queue = new Vector();

    setDistanceCost(endNode);

   queue.addElement(startNode);
   startNode.setTested(true);

   while(queue.size()>0){
     PathNode testNode = (PathNode)queue.firstElement();
     queue.removeElementAt(0);
     temp.addElement(testNode);

     if(testNode.getNodeID() == endNode.getNodeID()){
       pathToGoal.addElement(testNode);
       PathNode backNode = (PathNode)testNode.getBackTrace();
       pathToGoal.addElement(backNode);
       while(backNode.getBackTrace()!= null){
         backNode = (PathNode)backNode.getBackTrace();
         pathToGoal.addElement(backNode);
       }//

       return pathToGoal;
     }

     if(!testNode.expanded){
       testNode.expand(queue,PathNode.INSERT_COVER);
     }

   }//end while

   return null;

}//end coverfs


/**
 * paint() paints the searchgraph node by node
 */
```

```
  public void paint(Graphics g){
     Enumeration e = this.elements();
    while(e.hasMoreElements() ){
      PathNode nextNode = (PathNode)e.nextElement();
      nextNode.paint(g);
    }//end while
  }//end paint


}//end PathSearchGraph
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;


/**
 * class PathNodes creates nodes used to
 * establish a search graph network for the agent's
 * path planning
 */
public class PathNode extends Object {

  Vector links;       // holds links to other nodes

  int depth;          //depth in a tree from start node
  boolean expanded;   // indicates if node has been expanded
  boolean tested;     //indicates if node was ever tested
  int xPosition;      // x-coord of node
  int yPosition;      //y-coord of node
  String nodeID;      //string ID
  float distanceCost = 0;
  float coverBenefit = 0;
  PathNode backTrace = null;

  public static final int FRONT = 0;
  public static final int BACK = 1;
  public static final int INSERT_DIST = 2;
  public static final int INSERT_COVER = 3;

  public PathNode(int x, int y, float c) {
    xPosition = x;
    yPosition = y;
    nodeID = "" + (1000*x + y);
    coverBenefit = c;
    depth = 0;
    links = new Vector();
    expanded = false;
    tested = false;
```

```
}//end constructor


public int getX(){ return xPosition;}
public int getY(){ return yPosition;}
public String getNodeID(){ return nodeID;}
public float getCover() { return coverBenefit;}
public float getDistance() { return distanceCost; }
public boolean isleaf() { return (links.size() ==0) ; }
public PathNode getBackTrace(){ return backTrace;}

public void setDistanceCost(float d){  distanceCost = d; }
public void setDepth(int d) { depth = d; }
public void setExpanded(boolean state) { expanded = state;}
public void setTested (boolean state) { tested = state; }

public void reset(){
  depth = 0;
  distanceCost = 0;
  expanded = false;
  tested = false;
  backTrace = null;
}//end reset

public void setBackTrace(PathNode pn){
  backTrace = pn;
}

public void addLink(PathNode pn){
   links.addElement(pn);

}//end addLink


public void expand(Vector queue, int position){
  setExpanded(true);
  for(int j = 0; j < links.size(); j++){
    PathNode nextNode = (PathNode)links.elementAt(j);
    if(!nextNode.tested){

      nextNode.setTested(true);
      nextNode.setBackTrace(this);
      nextNode.setDepth(depth + 1);

      switch(position) {
        case FRONT:  queue.insertElementAt(nextNode,0);
        break;

        case BACK: queue.addElement(nextNode);
        break;

        case INSERT_DIST:
          boolean inserted = false;
          float nextDistanceCost = nextNode.getDistance();
```

```
            for(int k = 0; k < queue.size(); k++){
              //find where to insert node
              if(nextDistanceCost < ((PathNode)queue.elementAt(k)).getDistance()){
                queue.insertElementAt(nextNode, k);
                inserted = true;
                break;  //exit for loop
              }//end if

            }//end for

            //couldn't find place to insert, so add to end of queue
            if(!inserted) queue.addElement(nextNode);
          break;

          case INSERT_COVER:
            boolean isInserted = false;
            float nextCover = nextNode.getCover();


            for(int k = 0; k < queue.size(); k++){
              //find where to insert node
              if(nextCover > ((PathNode)queue.elementAt(k)).getCover()){
                queue.insertElementAt(nextNode, k);
                isInserted = true;
                break;  //exit for loop
              }//end if

            }//end for

            //couldn't find place to insert, so add to end of queue
            if(!isInserted) queue.addElement(nextNode);
          break;


        }//end switch(position)
      }
    }//end for
  }//end expand


  public void paint(Graphics g){
    g.setColor(Color.white);
    g.fillRect((getX()/Parameters.CELL_WIDTH)*Parameters.CELL_WIDTH,
      (getY()/Parameters.CELL_HEIGHT)*Parameters.CELL_HEIGHT,
          Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
    g.setColor(Color.red);
    Enumeration e = links.elements();
    while(e.hasMoreElements()){
      PathNode linkNode = (PathNode)e.nextElement();
      g.drawLine(getX(),getY(),linkNode.getX(),linkNode.getY());
    }
  }//end paint


}//end PathNode
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */


package thesis;

/**
 * class PathElement creates path objects that are
 * created to serve as navigation points for the agents to follow
 */

public class PathElement {
 double xVal = -1;
 double yVal = -1;
 int direction = -1;

  public PathElement(double x, double y) {
     xVal = x;
     yVal = y;
  }

  public double getX(){
     return xVal;
  }

  public double getY(){
     return yVal;
  }

  public int getD(){
     return direction;
  }
}//end PathElement
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;


/**
 * class Infrastructure  creates a series of building objects
 * that serve as the infrastructure for the urban environment
 *
 */
public class Infrastructure {
  Building1 b1a,b1b,b1c,b1d,b1e;
  Building2 b2a,b2b,b2c;

  /**
   * Infrastucture constructor
   */
  public Infrastructure() {
    b1a = new Building1(50,250);
    b1b = new Building1(45,400);
    b1c = new Building1(200,400);
    b1d = new Building1(355,400);

    b2a = new Building2(200,75);
    b2b = new Building2(475,75);
    b2c = new Building2(475,240);
  }//end constructor


  /**
   * paint()
   * paints the infrastructure building by building
   */
  public void paint(Graphics g){
     //draw buildings
     b1a.paint(g);
     b1b.paint(g);
     b1c.paint(g);
```

```
    b1d.paint(g);
    b2a.paint(g);
    b2b.paint(g);
    b2c.paint(g);
  }//end paint

}//end infrastructure
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 *
 **/


 package thesis;


 /**
  * class Building
  * Description:   Parent class to building types
  */

public class Building {

  int xPosition = 0;  //x coordinate of building
  int yPosition = 0;  //y coordinate of building
  int hWall;        //size of horizontal walls in environment grid squares
  int vWall;        //size of vertical walls
  int hDoor;        //size of horizontal doors
  int vDoor;        //size of vertical doors


  /**
   * Building() constructor
   */
  public Building() {
    hWall = Parameters.WALL_S;
    vWall = Parameters.WALL_S;
    hDoor = Parameters.DOOR_S;
    vDoor = Parameters.DOOR_S;
  }



/**
 * set() methods
 * sets attribute to parameter value
 */

  //sets x coordinate of building to x
  public void setX(int x){
    xPosition = x;
  }

  //sets y coordinate of building to y
  public void setY(int y){
      yPosition = y;
  }
```

```
 //sets horizontal wall size to hw
 public void setHWall(int hw){
   hWall = hw;
 }

//sets vertical wall size to vw
public void setVWall(int vw){
   vWall = vw;
}

//sets horizontal door size to hd
public void setHDoor(int hd){
   hDoor = hd;
}

//sets vertical door size to vd
public void setVDoor(int vd){
   vDoor = vd;
}

 /**
  * get() methods
  */

  //returns x coordinate
public int getX(){
   return xPosition;
}

//returns y coordinate
public int getY(){
   return yPosition;
}

//returns horizontal wall size
public int getHWall(){
   return hWall;
}

//returns vertical wall size
public int getVWall(){
   return vWall;
}

//returns horizontal door size
public int getHDoor(){
   return hDoor;
}

//returns vertical door size
public int getVDoor(){
   return vDoor;
}

}//end building
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *             in an urban environment
 * Copyright:   Copyright (c) 2001
 * Company:     USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;


 /**
  * class Building1
  * Description:  child class of Building
  * Creates a simple 4 wall building with 4 doors
  */

public class Building1 extends Building{

  Vector buildingComponents; //holds building components: doors and walls
  Wall tempWall;
  Door tempDoor;

  /**
   * Building1 constructor creates Wall and Door objects
   * and adds them to buildingComponent Vector
   * @param x x-coordinate of upper left corner of building
   * @param y y-coordinate of upper left corner of building
   */
  public Building1(int x, int y) {
    super.setX(x);
    super.setY(y);
    buildingComponents = new Vector();

    //north wall
    tempWall = new Wall(getX(),
                  getY(),
                  getHWall(),
                  Parameters.HORIZONTAL);
    buildingComponents.addElement(tempWall);

    tempDoor = new Door(getX() +Parameters.CELL_WIDTH*(getHWall()),
                  getY(),
                  getHDoor(),
```

```
                  Parameters.HORIZONTAL,
                  Parameters.DOWN);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX() +
                     Parameters.CELL_WIDTH *(getHWall() + getHDoor()),
                  getY(),
                  getHWall(),
                  Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);


//south wall
tempWall = new Wall(getX(),
                  getY()+ Parameters.CELL_HEIGHT*
                        (2*getVWall() + getVDoor()+1),
                  getHWall(),
                  Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);

tempDoor = new Door(getX() + Parameters.CELL_WIDTH * getHWall(),
                  getY()+ Parameters.CELL_HEIGHT*
                        (2*getVWall() + getVDoor()+1),
                  getHDoor(),
                  Parameters.HORIZONTAL,Parameters.UP);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX() + Parameters.CELL_WIDTH *
                        (getHWall() + getHDoor()),
                  getY()+ Parameters.CELL_HEIGHT*
                        (2*getVWall() + getVDoor()+1),
                  getHWall(),
                  Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);



//west wall
tempWall = new Wall(getX(),
                  getY()+ Parameters.CELL_HEIGHT *(1),
                  getVWall(),
                  Parameters.VERTICAL);
buildingComponents.addElement(tempWall);

tempDoor = new Door(getX(),
                  getY()+ Parameters.CELL_HEIGHT *
                         (getVWall() + 1),
                  getVDoor(),
                  Parameters.VERTICAL,Parameters.RIGHT);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX(),
                  getY()+ Parameters.CELL_HEIGHT *
                              (getVWall() + 1 + getVDoor()),
                  getVWall(),
                  Parameters.VERTICAL);
```

```
    buildingComponents.addElement(tempWall);


    //east wall
    tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                            (2*getHWall() + getHDoor()-1) ,
                      getY()+ Parameters.CELL_HEIGHT *(1),
                      getVWall(),
                      Parameters.VERTICAL);
    buildingComponents.addElement(tempWall);

     tempDoor = new Door(getX()+ Parameters.CELL_HEIGHT *
                            (2*getHWall() + getHDoor() - 1) ,
                      getY()+ Parameters.CELL_HEIGHT *
                           (getVWall() + 1),
                      getVDoor(),
                      Parameters.VERTICAL,Parameters.LEFT);
    buildingComponents.addElement(tempDoor);

    tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                            (2*getHWall() + getHDoor() - 1) ,
                      getY()+ Parameters.CELL_HEIGHT *
                            (getVWall() + 1 + getVDoor()),
                      getVWall(),
                      Parameters.VERTICAL);
    buildingComponents.addElement(tempWall);

  }//end constructor


  /**
   * paint() paints the instantiation of Building1
   * by call each Wall and Door objects' paint() function
   */
  public void paint(Graphics g){
    BuildingComponent tempComponent;
    for(Enumeration e = buildingComponents.elements();
        e.hasMoreElements(); ){
         tempComponent = (BuildingComponent) e.nextElement();
         tempComponent.paint(g);
    }//end for loop
  }//end paint

}//end building1
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

 package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

 /**
  * class Building2
  * Description:  child class of Building
  * creates a 2-room building with 4 horizontal doors
  * and 3 vertical doors
  */
public class Building2  extends Building{

  Vector buildingComponents; //holds building components: doors and walls
  Wall tempWall;
  Door tempDoor;


  /**
   * Building2 constructor creates Wall and Door objects
   * and adds them to buildingComponent Vector
   * @param x x-coordinate of upper left corner of building
   * @param y y-coordinate of upper left corner of building
   */
  public Building2(int x, int y) {
    super.setX(x);
    super.setY(y);

    buildingComponents = new Vector();
    //north wall
    tempWall = new Wall(getX(),
                  getY(),
                  getHWall(),
                  Parameters.HORIZONTAL);
    buildingComponents.addElement(tempWall);

    tempDoor = new Door(getX() +Parameters.CELL_WIDTH*getHWall(),
                  getY(),
                  getHDoor(),
                  Parameters.HORIZONTAL,
```

```
                 Parameters.DOWN);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX() +
                     Parameters.CELL_WIDTH *(getHWall() + getHDoor()),
                 getY(),
                 getHWall(),
                 Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);


//north wall#2
tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                             (2*getHWall() + getHDoor() ) ,
                 getY(),
                 getHWall(),
                 Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);

tempDoor = new Door(getX()+ Parameters.CELL_HEIGHT *
                             (3*getHWall() + getHDoor() ) ,
                 getY(),
                 getHDoor(),
                 Parameters.HORIZONTAL,
                 Parameters.DOWN);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                             (3*getHWall() + 2*getHDoor() ) ,
                 getY(),
                 getHWall(),
                 Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);


//south wall
tempWall = new Wall(getX(),
                 getY()+ Parameters.CELL_HEIGHT*
                       (2*vWall + getVDoor()+1),
                 getHWall(),
                 Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);

tempDoor = new Door(getX() + Parameters.CELL_WIDTH * getHWall(),
                 getY()+ Parameters.CELL_HEIGHT*
                       (2*getVWall() + getVDoor()+1),
                 getHDoor(),
                 Parameters.HORIZONTAL,Parameters.UP);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX() + Parameters.CELL_WIDTH *
                      (getHWall() + getHDoor()),
                 getY()+ Parameters.CELL_HEIGHT*
                       (2*getVWall() + getVDoor()+1),
                 getHWall(),
                 Parameters.HORIZONTAL);
```

```
buildingComponents.addElement(tempWall);

//south wall #2
tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                    (2*getHWall() + getHDoor() ) ,
          getY()+ Parameters.CELL_HEIGHT*
                (2*getVWall() + getVDoor()+1),
          getHWall(),
          Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);

tempDoor = new Door(getX()+ Parameters.CELL_HEIGHT *
                    (3*getHWall() + getHDoor() ) ,
          getY()+ Parameters.CELL_HEIGHT*
                (2*getVWall() + getVDoor()+1),
          getHDoor(),
          Parameters.HORIZONTAL,Parameters.UP);
buildingComponents.addElement(tempDoor);


tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                    (3*getHWall() + 2*getHDoor() ) ,
          getY()+ Parameters.CELL_HEIGHT*
                (2*getVWall() + getVDoor()+1),
          getHWall(),
          Parameters.HORIZONTAL);
buildingComponents.addElement(tempWall);

//west wall
tempWall = new Wall(getX(),
          getY()+ Parameters.CELL_HEIGHT *(1),
          getVWall(),
          Parameters.VERTICAL);
buildingComponents.addElement(tempWall);

tempDoor = new Door(getX(),
          getY()+ Parameters.CELL_HEIGHT *
                 (getVWall() + 1),
          getVDoor(),
          Parameters.VERTICAL,Parameters.RIGHT);
buildingComponents.addElement(tempDoor);

tempWall = new Wall(getX(),
          getY()+ Parameters.CELL_HEIGHT *
                    (getVWall() + 1 + getVDoor()),
          getVWall(),
          Parameters.VERTICAL);
buildingComponents.addElement(tempWall);

//mid wall
tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                    (2*getHWall() + getHDoor() - 1) ,
          getY()+ Parameters.CELL_HEIGHT *(1),
          getVWall(),
          Parameters.VERTICAL);
buildingComponents.addElement(tempWall);
```

```
     tempDoor = new Door(getX()+ Parameters.CELL_HEIGHT *
                              (2*getHWall() + getHDoor() - 1) ,
                   getY()+ Parameters.CELL_HEIGHT *
                          (getVWall() + 1),
                   getVDoor(),
                   Parameters.VERTICAL,Parameters.RIGHT);
    buildingComponents.addElement(tempDoor);

    tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                              (2*getHWall() + getHDoor() - 1) ,
                   getY()+ Parameters.CELL_HEIGHT *
                              (getVWall() + 1 + getVDoor()),
                   getVWall(),
                   Parameters.VERTICAL);
    buildingComponents.addElement(tempWall);

    //eastwall
     tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                              (4*getHWall() + 2*getHDoor() - 1) ,
                   getY()+ Parameters.CELL_HEIGHT *(1),
                   getVWall(),
                   Parameters.VERTICAL);
    buildingComponents.addElement(tempWall);

     tempDoor = new Door(getX()+ Parameters.CELL_HEIGHT *
                              (4*getHWall() + 2*getHDoor() - 1) ,
                   getY()+ Parameters.CELL_HEIGHT *
                          (getVWall() + 1),
                   getVDoor(),
                   Parameters.VERTICAL,Parameters.LEFT);
    buildingComponents.addElement(tempDoor);

    tempWall = new Wall(getX()+ Parameters.CELL_HEIGHT *
                              (4*getHWall() + 2*getHDoor() - 1) ,
                   getY()+ Parameters.CELL_HEIGHT *
                              (getVWall() + 1 + getVDoor()),
                   getVWall(),
                   Parameters.VERTICAL);
    buildingComponents.addElement(tempWall);
  }//end constructor

  /**
   * paint() paints the instantiation of Building2
   * by call each Wall and Door objects' paint() function
   */
  public void paint(Graphics g){
    BuildingComponent tempComponent;
    for(Enumeration e = buildingComponents.elements();
        e.hasMoreElements(); ){
         tempComponent = (BuildingComponent) e.nextElement();
         tempComponent.paint(g);
    }//end for loop
  }//end paint
}//end building2
```

```
/**
 * Title:       Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *              in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;


 /**
  * class BuildingComponent
  * Description:  parent class of objects
  * used to create buildings:  Walls and Doors
  */

 public class BuildingComponent extends SimObject{

  //orientation of building component:
  //Parameters.HORIZONTAL or Parameters.VERTICAL
  int orientation = 0;

  //component size in grid squares
  int size = 0;

  //constructor
  public BuildingComponent() {
  }


  /**
   * set() methods
   */

   //sets size of component to s
   public final void setSize(int s){
     size = s;
  }

  //sets orientation to o
  public final void setOrientation(int o){
     orientation = o;
  }
```

```
 /**
  * get() methods
  */

  //returns component size
 public final int getSize(){
    return size;
 }

 //returns component orientation
 public final int getOrientation(){
   return orientation;
 }

  /**
  * paint() function
  * paints component to screen
  */
  public void paint(Graphics g){
    g.fillRect(getX(),getY(),
           Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
 }//end paint


}//end BuildingComponent
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

/**
 * class Door
 * Description:  child class of BuildingComponent
 * creates a door
 */

public class Door extends BuildingComponent{

  int inDirection = 0;

  /**
   * Door constructor
   * @param x x-coordinate
   * @param y y-coordinate
   * @param s size of door
   * @param o orientation of door
   * @param i entrance direction
   */

  public Door(int x, int y, int s, int o, int i) {

     super.setType(Parameters.DOOR);
     super.setXpos(x);
     super.setYpos(y);
     super.setSize(s);
     super.setOrientation(o);
     inDirection = i;
     int cover1, cover2;

     if(getOrientation() == Parameters.HORIZONTAL){

        //inserts a type value of Parameters.Door
        //in the SIM_ENV array for each grid square the door
        //occupies
```

```
    for(int ix = 0; ix < getSize(); ix ++){
      Parameters.SIM_ENV[y/Parameters.CELL_HEIGHT]
                [x/Parameters.CELL_WIDTH + ix] = Parameters.DOOR;
    }

    //creates a cover attribute
    //10 for inside the door; 0 for outside
    if(inDirection == Parameters.DOWN){
      cover1 = 10;
      cover2 = 0;
    }
    else{
      cover1 =0;
      cover2=10;
    }

    //creates 2 nodes associated with the door
    //one node for inside the door
    //one node for outside the door
    PathNode newNode = new PathNode(x+s/2*Parameters.CELL_WIDTH,
                        y+30, cover1);
    Parameters.pathGraph.put(newNode);
    newNode = new PathNode(x+s/2*Parameters.CELL_WIDTH, y-30, cover2);
    Parameters.pathGraph.put(newNode);
  }
  else{

     //inserts a type value of Parameters.Door
     //in the SIM_ENV array for each grid square the door
     //occupies
     for(int ix = 0; ix < getSize(); ix ++){
      Parameters.SIM_ENV[y/Parameters.CELL_WIDTH + ix]
                 [x/Parameters.CELL_HEIGHT] = Parameters.DOOR;
     }

    //creates a cover attribute
    //10 for inside the door; 0 for outside
    if(inDirection == Parameters.RIGHT){
      cover1 = 10;
      cover2 = 0;
    }
    else{
      cover1 =0;
      cover2=10;
    }

    //creates 2 nodes associated with the door
    //one node for inside the door
    //one node for outside the door
    PathNode newNode = new PathNode(x+30,
                        y+s/2*Parameters.CELL_WIDTH,cover1);
    Parameters.pathGraph.put(newNode);
    newNode = new PathNode(x-30, y+s/2*Parameters.CELL_WIDTH, cover2);
    Parameters.pathGraph.put(newNode);

  }
```

```
  }//end constructor


  /**
   * getInDirection()  returns the Door object's entrance direction
   * @return inDirection
   */
  public int getInDirection(){
    return inDirection;
  }

  /**
   * paint() paints the door in gray
   */
  public void paint(Graphics g){
    g.setColor(Color.gray);
    for(int ix = 0; ix < size; ix ++){
        if(orientation == Parameters.HORIZONTAL){
          g.drawRect(getX()+(ix*Parameters.CELL_WIDTH),getY(),
            Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
        }
        else{
           g.drawRect(getX(),getY()+ (ix*Parameters.CELL_HEIGHT),
            Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
        }//end if/else
    }//end for
  }//end paint


}//end Door
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;
import java.awt.image.*;
import java.lang.*;
import java.io.*;
import java.net.*;
import javax.swing.*;

/**
 * class Wall
 * Description: child of BuildingComponent
 * creates a wall
 */

public class Wall extends BuildingComponent {


/**
 * Wall constructor
 * creates a wall at (x,y)
 * with a size s and orientation o
 * @param x Wall object x-coord
 * @param y Wall object y-coord
 */
  public Wall(int x, int y, int s, int o) {
      super.setType(Parameters.WALL);
      super.setXpos(x);
      super.setYpos(y);
      size = s;
      orientation = o;
      if(getOrientation() == Parameters.HORIZONTAL){
        for(int ix = 0; ix < getSize(); ix ++){
          Parameters.SIM_ENV[y/Parameters.CELL_HEIGHT]
                    [x/Parameters.CELL_WIDTH + ix] = Parameters.WALL;
        }
     }
     else{
         for(int ix = 0; ix < getSize(); ix ++){
          Parameters.SIM_ENV[y/Parameters.CELL_WIDTH + ix]
                      [x/Parameters.CELL_HEIGHT] = Parameters.WALL;
        }
     }
  }//end constructor
```

```
/**
 * paint() paints the Wall object
 */
  public void paint(Graphics g){
     g.setColor(Color.gray);
     for(int ix = 0; ix < size; ix ++){
          if(orientation == Parameters.HORIZONTAL){
            g.fillRect(getX()+(ix*Parameters.CELL_WIDTH),getY(),
              Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
          }
          else{
             g.fillRect(getX(),getY()+ (ix*Parameters.CELL_HEIGHT),
              Parameters.CELL_WIDTH,Parameters.CELL_HEIGHT);
          }//end if/else
     }//end for
  }//end paint


}//end Wall
```

```
/**
 * Title:        Urban Combat Simulation
 * Description:  An intelligent agent simulation of a squad
 *               in an urban environment
 * Copyright:    Copyright (c) 2001
 * Company:      USMC
 * @author Capt Arthur R. Aragon
 * @version 1.0
 */

package thesis;
import java.awt.*;

/**
 * class SimObject
 * Description: parent class of all simulation objects
 */
public class SimObject {

  int xPosition;  //int between 0 and maxX
  int yPosition;  //int between 0 and maxY
  int type;

  //constructor
  public SimObject() {
  }

   /**
   * void setXpos(int x)
   * sets agent's xPosition to x
   */
  public final void setXpos(int x){
     xPosition = x;
  }

  /**
   * void setYpos(int y)
   * sets agent's yPosition to y
   */
  public final void setYpos(int y){
     yPosition = y;
  }

  /**
   * void setType(int t)
   */
   public final void setType(int t){
     type = t;
   }

   /**
   * int getX()
   * returns agent's xPosition
   */
  public final int getX(){
     return xPosition;
```

```
  }

  /**
   * int getY()
   * returns agent's yPosition
   */
  public final int getY(){
     return yPosition;
  }

  /**
   * int getType()
   */
   public final int getType(){
      return type;
   }


}//end SimObject
```

LIST OF REFERENCES

Borland®, Learning Java ® with JBuilder™: JBuilder 4. Computer software. Inprise Corporation. 2000

Burbeck, Steve. "Applications Programming in Smalltalk-80™: How to use Model-View-Controller (MVC)". Copyright 1987

Bigus, Joseph P, and Jennifer Bigus. Constructing Intelligent Agents with Java™. John Wiley & Sons, Inc. 1998.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. Introduction to Algorithms. The MIT Press. McGraw-Hill Book Company. 1990.

Dietel, H.M. and P.J. Dietel. Java™ How To Program. Third Edition. Prentice Hall. 1999.

Eriksson, Hans-Erik and Magnus Penker. UML Toolkit. John Wiley & Sons, INC. 1998.

Hofmeister, Christine, Robert Nord, and Dilip Soni. Applied Software Architecture. Addison Wesley Longman, Inc. 2000.

Ilachinski, A., Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare (U), (Center for Naval Analyses Research Memorandum CRM 97-61.10), Alexandria, VA: Center for Naval Analyses, 1997.

Koosis, Donald and David Koosis. Java™ Programming For Dummies,. 3rd Edition. IDG Books WorldWide, Inc. 1998.

Rothenberg, Jeff. "Object-Oriented Simulation: Where Do We Go from Here?" A RAND NOTE. The Rand Corporation. October 1989.

Rothenberg, Jeff. "The Nature of Modeling." A RAND NOTE. The Rand Corporation. November 1989.

UNEP United Nations Population Division "GEO- 2000 Global Environment Outlook", Chapter Two: The State of the Environment. Geneva 1997.

United Nations Population Division, "World Urbanization Prospects: The 1994 Revision", UN New York, 1995.

Weiss, Gerhard. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press. Cambridge, Massachusetts. 1999

THIS PAGE INTENTIONALLY LEFT BLANK

BIBLIOGRAPHY

Cellier, Francois E. Progress in Modeling and Simulation. Academic Press. 1982.

Luger, George F., and William A. Stubblefield. Artificial Intelligence: Structures and Strategies for Complex Problem Solving. Addison Wesley Longman, Inc. 1998.

Milton, T.R. Jr., "Urban Operations: Future War," Military Review, 1 February 1994.

Neelamkavil, Francis. Computer Simulation and Modeling. John Wiley & Sons Ltd. 1987.

Negoita, Constantin V. and Dan Ralescu. Simulation, Knowledge-Based Computing, and Fuzzy Statistics. Van Nostrand Reinhold Company Inc. 1987.

Peters, James F. and Witold Pedrycz. Software Engineering: An Engineering Approach. John Wiley & Sons, Inc. 2000.

Spriet Jan A. and Ghislain C. Vansteenkiste. Computer-aided Modeling and Simulation. Academic Press Inc. 1982.

Ripley, Brian D. Stochastic Simulation. John Wiley & Sons Ltd. 1987.

United Nations Environmental Program (UNEP), *The Global Environmental Outlook* (draft), Nairobi, 1996

United States Army, Field Manual 90-10-1: An Infantryman's Guide To Urban Combat (Washington D.C.: Department of Defense, 1979).

Uttamsingh, Ranjeet J. and A. Martin Wildberger. Artificial Intelligence and Simulation. Simulation Series. Volume 23 Number 4. Simulation Councils, INC. 1991.

Zobrist, George W. and James V. Leonard. Object-Oriented Simulation: Reusability, Adaptability, Maintainability. IEEE PRESS. 1997.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California
   debarber@nps.navy.mil

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia
   webmaster@tecom.usmc.mil

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia
   ramkeyce@tecom.usmc.mil
   strongka@tecom.usmc.mil
   sanftlebenka@tecom.usmc.mil

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California
   doranfv@mctssa.usmc.mil
   palanaj@mctssa.usmc.mil

7. Director, Studies and Analysis Division, MCCDC, Code C45
   Quantico, Virginia
   thesis@mccdc.usmc.mil

8 Dr. Neil Rowe, Code32
   Computer Science Department
   Naval Postgraduate School
   rowe@cs.nps.navy.mil

9. CDR Chris Eagle, Code32
   Military Instructor, Computer Science Department
   Naval Postgraduate School
   cseagle@cs.nps.navy.mil