SIMPLE MULTI-AGENT COOPERATION:
AN APPROACH BASED ON PREDATOR-PREY MODELING

By

RICHARD J. BARTLESON JR.

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2003

This thesis is dedicated to my late wife, Candace D. Bartleson. The completion of my graduate studies and this thesis would not have been possible without her unswerving support and encouragement. Her memory continues to inspire me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

SIMPLE MULTI-AGENT COOPERATION: AN APPROACH BASED ON PREDATOR-PREY MODELING

By

Richard J. Bartleson Jr.

May 2003

Chair: Douglas D. Dankel II
Major Department: Computer and Information Science and Engineering

This thesis focuses on using simple rules for cooperation within a multi-agent system based on a predator-prey model. Agents are software entities that function somewhat independently within an artificial world, and a multi-agent system is a software program that allows multiple agents to coordinate their efforts. The predator-prey model used in the context of this thesis is based loosely on a pride of lions while hunting prey.

For the computational portion of this thesis, a predator-prey program was developed from an object-oriented design based on a simple predator-prey model. The design includes an environment class for the artificial world in the model, a plant class to represent the food source for the prey, and a creature class that is sub-classed into predator and prey classes to represent the animals. Two predator modes were used: a simple pursuit mode and a flank and converge mode. In each mode, a lead predator chooses the target prey and the target is pursued until it is either captured or it escapes.

The efficiency and effectiveness of the two predator modes were compared on the basis of number of turns taken during a prey pursuit, whether or not the prey was captured or escaped, and the distance the prey traveled while being pursued. The flank and converge mode showed marked improvement over the simple pursuit mode.

CHAPTER 1
INTRODUCTION

This thesis examines using simple rules for agent cooperation within a multi-agent system. In this context, agents are software entities that exist in a created, artificial environment. They can manipulate their environment in some fashion while pursuing a goal using a pre-specified set of capabilities. A multi-agent system is a collection of agents that exist together in the same environment, which may or may not cooperate directly with each other.

The cooperation among agents in a multi-agent system is seen as a form of coordination between non-antagonistic agents that may share the same goal. When pursuing the same goal, the existence of multiple agents allows the tasks for achieving the goal to be divided among the agents. The capacity for task division is an essential part of the multi-agent system.

Beyond the capacity for task division, a multi-agent system has the capability of information sharing among the agents. Information sharing is required when they are trying to achieve a goal that would be impossible or extremely difficult to achieve individually. But, the sharing of information is not free as there are costs associated with generating, distributing, and managing the information.

Multi-agent systems have been used effectively in many real world activities. These activities include industrial applications such as product design, planning activities, and real-time control. In the industrial domain, the agents usually represent various components that exist in the real world but are difficult to model without the use of a computer. At least one non-industrial area for multi-agent usage has been examined in the form of directed improvisation. In directed improvisation, the agents take on the role of characters acting out user-specified directions in an improvised setting.

For this thesis, the agents are entities in a predator-prey model partially based on animal cooperation, particularly the type exhibited by lions. When pursuing a shared goal, animal cooperation occurs when the outcome for the group of cooperating animals exceeds the collective costs to the individual animals within the group. For lions, the ability to cooperate in hunting is well documented and serves as the basic model for the predator-prey model used in the software developed for study.

The predator-prey model uses an artificial environment in which all agent activities occur. There is a food source for the prey to consume that requires the prey to move around in the environment, subjecting them to possible pursuit by the predators. The predators have the goal of capturing the prey but have to work together to do so. The cooperation among the predators can be achieved with a set of simple rules.

The predator-prey program written for this thesis was developed using an object-oriented design. The various classes of objects include an environment class to represent the artificial world in the model, a plant class to represent the food source for the prey, and a creature class that is sub-classed into predator and prey classes to represent the animals. The program is a turn-based simulation in which the prey and predators perform individualized activities during various turns based on a speed parameter assigned to each creature in the environment. The activities of the prey include moving from plant to plant for food consumption and evading predators when they are near. The predators travel around the environment hunting for prey to pursue and capture.

The chapters that follow include a literature review of multi-agent systems and cooperation among animals, a discussion of the approach used to develop a predator-prey program, the results of executing the program, and concluding remarks with proposed future work.

CHAPTER 2
LITERATURE REVIEW

This chapter provides a review of literature related to multi-agent systems and cooperation in animals, particularly lions. Since the focus of this thesis is simple cooperation rules within a multi-agent environment as modeled by a predator-prey scenario loosely based on cooperative hunting within a pride of lions, this chapter contains two major section: a section on agents and multi-agent systems, and a section covering animal cooperation in general and lions in particular.

The first section begins by presenting a definition of an agent and then discusses multi-agent environments and their uses. This includes showing how they have been used to solve real-world problems and to simulate possible scenarios with real-world complexities. The second section covers the basics of animal cooperation with examples across various species and then discusses cooperative behaviors of male and female lions within a pride.

Overview of Multi-Agent Systems

What Are Agents?

There are varied descriptions and perceptions of what constitutes an agent. These views range from simple entities to intelligent ones, all of which have varying degrees of influence on their surrounding environment as well as various reactions to stimuli within their environment.

One view is that an agent is a computer system or program situated within some environment that is capable of producing independent action within that given environment. Further, agents can perform these actions autonomously to meet their design objectives [Woo99, 29].

The design objectives for an agent are developed based on what the agent is supposed to accomplish relative to the properties of its environment. The properties of a given environment can be described along the following scales:

- Accessible vs. Inaccessible – the degree of information available about the environment.
- Deterministic vs. Non-deterministic – the relationship between action and effect within the environment.
- Episodic vs. Non-episodic – the amount of interaction between scenario sequences of agents in the environment.
- Static vs. Dynamic – the degree to which the agent influences the environment.
- Discrete vs. Continuous – the number of actions and percepts of the agent within the environment [Woo99, 30-31].

Agents are given design objectives based on their intended operation within and in response to the varied properties of their environment. And, in some cases, agents are able to influence their environment.

Another more complex form of agent is called an intelligent agent. An *intelligent agent* is an agent with the additional capability of *flexible* autonomous action to meet its design objectives. Within this context, flexible means reactivity, pro-activeness, and social ability. Reactivity is the ability to perceive and respond to the environment. Pro-activeness is the ability to be goal-directed or to take the initiative. Social ability refers to the capacity to interact with other agents in a cooperative manner [Woo99, 32].

Another characterization of agents sees an agent as "extracting information about the world it operates in via its sensors" [Smi95, 151] and then uses this information to construct internal models of its world and perform some decision-making process in pursuit of a goal. This particular view equates agents to information processing systems wherein the agent experiences an “interaction space” resulting from its interaction with its environment.

Also, agents are sometimes viewed as being alive in a manner of speaking. Kennedy and Eberhart say, “The most interesting thing to us about autonomous agents is the tendency we have to anthropomorphize their behavior” [Ken01, 129], thereby giving them sensations, actions, understanding, and agendas. An agent can be seen as having abstractions such as “intentions and know-how” [Sin94, 3]. Fundamentally, we can speak of agents in a sense as being alive inside their environment.

The preceding descriptions and discussion of what an agent is can be summarized by a definition. As given by Tessier, Muller, Fiorino, and Chaudron [Tes01, 2], “An agent is a system with the following properties: 1. It lives in an artificial world W together with other agents, 2. It has facilities to sense W and to manipulate W, 3. It has a (at least partial) representation of W, 4. It is goal-directed, and as a consequence it has the ability to plan its activities, 5. It can communicate with other agents.”

In essence, agents can be thought of as semi-independent entities existing within a given, bounded environment that are capable of interacting with the environment in some specific manner. They are programmed to have a specific goal and have some set of capabilities for achieving the goal. An agent may manipulate the environment and be changed by the environment as a result of its actions.

Beyond the idea of individual agents working on an individual goal is the concept of a system involving multiple agents working together. In these systems, agents can choose to cooperate with each other in pursuit of their individual goals. These systems are referred to as multi-agent systems and involve substantial interaction among the agents, not just the interaction of an agent with the environment.

Coordination and Cooperation Among Agents

In a multi-agent system, agents are designed to interact with each other: sometimes toward the same goal and sometimes with separate goals. The system provides the context and infrastructure, and the cooperation achieved by the agents is usually accomplished through a combination of task allocation and information sharing.

The characteristics of a multi-agent environment include: 1) an infrastructure specifying protocols for communication and interaction, 2) openness and no centralized designer, and 3) self-interested or cooperative agents that are autonomous and distributed [Huh99, 81-82]. The infrastructure serves as part of the environment, providing a framework for the expression of each agent's capabilities. The openness and lack of a centralized designer allow a variety of interactions to emerge from the cooperative nature of the agents.

When agents work together in a given environment, there needs to be a means of coordinating their efforts. According to Huhns and Stephens [Huh99, 83], "Coordination is the property of a system of agents performing some activity in a shared environment." Furthermore, cooperation is the form of coordination existing among agents that are not antagonistic.

For effective cooperation to take place, a multi-agent system of sufficient complexity requires agents that are not only capable of performing their individual tasks but also can effectively interact with other agents. This usually involves the use of protocols for enforcing the modularity of the multi-agent system. These protocols assist in dividing the internal workings of an agent from its interface with the rest of the environment [Sin94, 125].

Once protocols for communication are in place, the work of task division is an essential component of a multi-agent interaction. A common method for deriving a solution when solving problems or attempting to achieve some goal is the divide-and-conquer approach. Within the context of multi-agent systems, this usually means subdividing the problem space into a set of tasks that can be distributed among the agents involved. Each agent is then responsible for solving a smaller size sub-problem that contributes to the solution of the overall problem.

To address the area of task division, some early work related to multi-agent systems involved communicating problem solvers. It was found that the organization of the nodes, the problem solvers, would have improved performance by reducing the responsibilities given to each node. A balance had to be achieved between the degrees of local bias versus external bias of each node in the system. Too much overlap between the sub-problem areas led to redundant work. However, too little overlap led to increased processing costs. Nodes had a particular "interest area" from the sub-problem solution space. Also, each node in the system could have problem-solving roles such as "integrator," "specialist," and "middle manager" [Dur87, 1280].

Also, part of the system of the communicating problem solvers involved the use of a plan to represent the sequence of related activities. The plan indicated the specific role of a node during a certain interval of time. Nodes could then use the additional information to improve their individual coordination knowledge through a dynamic refinement of interest areas [Dur87, 1282].

Another area of coordination is decision-theoretical planning, which is a blending of classic planning assumptions with economic concepts of valuation for different resources. Essentially, the model of interaction used is a combination of two reasoning types: 1) the explicit reasoning and sequential nature of coordination that occurs in planning models; and 2) the economic reasoning and decision-theoretic trade-offs found in economic models. This allows agents to "represent their uncertainty about the behavior of other agents" and permits agents to gain insight into the resource valuations and course of action of other agents. The "economic behavior of agents can be directly related" to the decisions of the agents when considering domain level processes [Bou99, 35-36].

So, the tasks derived from the divide-and-conquer approach can be distributed based on the following summarized criteria [Huh99, 99]:

- Critical resources should not be overloaded.
- An agent should be assigned tasks based on matching capabilities with other agents.
- Agents with a wide view should assign tasks to other agents.
- Agents should have overlapping responsibilities to achieve coherence.
- Agents with spatial or semantic proximity should be assigned highly interdependent tasks to minimize communication and synchronization costs throughout the system.
- Tasks should be reassigned if necessary for completing urgent tasks in a timely manner.

Beyond the need for dividing tasks among the agents in a multi-agent system, there also is the need to share information. This need develops as part of the view of agents as information-processors. Through their interactions, agents create local, individualized viewpoints of the environment in which they exist and function. For cooperation to be successful, information must be shared among the agents.

Through the sharing of information, a multi-agent system is able to achieve a given objective when individual agents would be incapable of doing so. As expressed by Lander and Lesser [Lan97, 193-194]:

> An agent with only a local view of the search space cannot avoid producing subproblem solutions that conflict with other agents' solutions and cannot make intelligent decisions about managing conflicts that do occur. To overcome this problem, information sharing can be used to enhance the local perspectives of agents. A better understanding of the global situation leads to more intelligent and focused search and, ultimately, to higher-quality solutions and more efficient performance.

But, the sharing of information does not come freely. There are costs associated with the information-sharing process. These costs can be divided into five distinct areas [Lan97, 204]: 1) Generation, 2) Determination, 3) Transmission, 4) Translation, and 5) Local Management. Generation refers to the costs expended by the sending agent in creating the shareable information. Determination refers to the cost of determining what should be shared and at what point in time. Transmission is the cost associated with moving messages from one agent to another. Translation is related to costs of translating from a local to global format or from the language of the sending agent to the language of the receiving agent. Local management refers to the costs of managing the shared data and of determining the relevance of received information.

So, a collection of agents can be used in a coordinated manner to create a multi-agent system. The system can then accomplish a set of design objectives of its own, via the interaction of the agents within the system. When the coordination is non-antagonistic, the agents can cooperate, allowing a larger problem to be solved by allowing it to be broken into more manageable tasks or sub-problems. Also, cooperation permits the sharing of resources and information.

Usages of Multi-Agent Systems

Multi-agent systems are usually created in an effort to solve some type of problem in a computational manner. They are developed with the understanding that a set of agents is more effective or more efficient in reaching a solution to the given problem. Each agent of a multi-agent system attempts to do its part to solve the problem whether the agent has identical design objectives to its teammates or not.

What follows is a discussion of some of the ways in which multi-agent systems have been used to support real-world industrial settings including the design of products, the planning and scheduling of activities, and the control of real-time systems. A non-industrial area where the use of multi-agent systems was explored is directed improvisation.

Agent-based systems have supported industrial applications covering the entire life cycle of systems and products. They have been used effectively in [Par99, 385] "product design, process operation at the planning and scheduling level, and process operation at the lower level of real-time equipment control."

When used for product design, a multi-agent system can help teams of designers, who are usually in different locations, to develop the components and subsystems of a product. The agents can be used to represent the various components and tools involved in the product. Complex trade-offs are created as more characteristics are included in any given design. Through the use of agents, these complex trade-offs, such as weight versus power consumption, can be examined in a variety of combinations [Par99, 387-389].

For systems involved in planning and scheduling, the agents take varying roles such as resources representing real-world items like machines and operators or as tasks like work orders or as service providers for services like dispatching and managing. The process operations of a production shop can thereby be modeled. The agents can coordinate with each other via a mechanism known as a contract net, where each agent bids for various services or resources within the system. These types of multi-agent systems have been used successfully in supply chain management and logistics management systems [Par99, 392-395].

Real-time control via a multi-agent system is a little more involved. It requires a faster response time and the use of more "semantically constrained information," which are data having a limited scope in size or time relative to a given situation [Par99, 395]. An example of semantically constrained information is the set of data associated with a machined engine part as it moves along an assembly line. As the engine part moves along the line, it is transformed by various mechanisms: a hole is bored followed by the threading of the hole followed by the inserting of a bolt. The relevant data about the part only has meaning at certain points in time. Real-time oriented, agent-based systems have been used to control robots supporting shipbuilding, to manage climate control in buildings, to handle painting of vehicles in an automotive paint shop, and to handle the machining of complex mechanical parts [Par99, 395-396].

Another area where multi-agent systems have been used is in directed improvisation. Directed improvisation is work that is simultaneously created and performed while being constrained by user-specified directions. For example, the agents can take on the role of characters that are given abstract directions, either interactively or with a script. The agent-level characters then develop a collaborative behavior in line with the user's directions, but which also includes "artfully improvised" elements that can "surprise and engage" [Hay98, 141].

So, multi-agent systems can be used in industrial and non-industrial environments. The individual agents in the systems are given distinct properties and the collaborative nature of their interaction leads to solutions for the given problem domain. The systems can run independently with very little input or interactively, responding to directions from users.

Brief Overview of Lion Cooperation While Hunting

This section provides an overview of cooperation among animals and of lions in particular. The cooperative hunting style of lions can be loosing associated with agents having a specific goal, the prey, while existing in a bounded environment, a pride's territory, that includes other agents that may cooperate.

Animal cooperation is discussed first, covering some of the many types of cooperative interactions that occur. Next, the nature of cooperation in lions is discussed. This covers the differences between male and female members of a pride, contrasting the different motives and cooperative style of each gender.

Animal Cooperation

Cooperation among animals can usually be observed by simply sitting outside in a park or other nature setting. A group of squirrels can often be seen working together: one squirrel will forage while its companions remain aloft in the trees to warn of the approach of a prowling predator. Alternatively, smaller birds can be observed working together to ward off a member of a larger bird species.

Beyond these simple observations, a more formal view of cooperation is provided by the following definition given by Dugatkin [Dug97, 14]:

> *Cooperation* is an outcome that—despite *potential* relative costs to the individual—is "good" in some appropriate sense for the members of a group, and whose achievement requires collective action.

This definition can lead to various forms of cooperative behavior. The models of cooperation used within evolutionary biology can be grouped into four broad categories: 1) cooperation via kin selection, 2) cooperation via group selection, 3) cooperation via reciprocity, and 4) cooperation via byproduct mutualism. Cooperation via kin selection refers to the genetic basis for cooperation. It considers the effect of the genes not only on an individual but "most importantly those sharing genes that are identical by descent (i.e., kin)" [Dug97, 15]. Cooperation via group selection deals with individuals choosing to become a member of a particular group. The selection is based on the individual's value of the costs for belonging to the group relative to the benefits gained either from membership within the group or by the group's interaction with other groups. Cooperation via reciprocity takes into account the possibility of defection, or lack of cooperation, on the part of individual members of a group. Studies have usually involved the use of the prisoner's dilemma, where two or more individuals are given a choice between cooperating and defecting with different weighted rewards per individual given to each combination of choices. Cooperation via byproduct mutualism refers to a cooperative effort between entities where one entity provides the bulk, if not all, of the effort to obtain a gain from the other entity when the other entity does not perform any direct action yet benefits from the cooperation. [Dug97, 17-34]

There are numerous documented examples of cooperation within animal groups ranging from fish to birds to non-primate mammals to primates. Fish, in addition to the familiar schooling effect, perform such actions as nest raiding where a group of females attack the nest of a competing group and cooperative foraging where a group of larger fish work together to corner a set of smaller fish to feed on them. In birds, members of a given species, even when competing among themselves, will join together to protect a territory against a common enemy. Also, species of hawks and falcons perform cooperative hunting by pursuing a chosen prey and then sharing in the rewards of the capture. Non-primate mammals, such as impala, squirrels, and elephants also have cooperative behaviors. Impala will groom one another. Squirrels give alarm calls to each other. Elephants form very strong maternal family units that protect their young by forming circular formations and have coalitions of larger females who work together to defend against potential dangers. Primates exhibit a wide range of cooperative behaviors such as reciprocal grooming, exchange of food, hunting together, and scanning for predators [Dug97, 45-141].

Nature of Cooperation in Lions

Lions are perceived by many to be a highly cooperative species, with an intricate capacity for hunting individually or in groups. They also form groups of varying size and composition relative to their environment, which is composed of sources of food as well as potential competitors.

As a social species, lions develop a variety of strategies for cooperation. The cooperation is oriented toward protecting the young cubs as well as for hunting. They form prides of up to 18 related female adults and their dependent offspring together with one to nine adult males [Sch91, 699]. In the prides that have more than three male adults, the coalition of male adults is “always composed of close relatives” [Pac97, 57].

There are differences in the cooperative behavior of male versus female lions. The male lions display their “greatest capacity for teamwork” when they work together to defend the territory of the pride from invading males [Pac97, 55]. This is done primarily for reproductive success of the males within the coalition. It is this reproductive advantage that encourages non-related males to form coalitions. Although the females perform most of the hunting in a pride, the males do participate [Pac97, 54-57].

The males usually choose the behavior of “territorial maintenance,” which consists of patrolling to keep the territory secure, scent marking, and roaring. This serves to maintain the pride’s territory by discouraging rivals. But, the males must balance the costs and benefits, pitting cub survival versus their own physical condition. Males have to select between accompanying the pride females and scavenging from them or hunting independently. Even when successful, males supplement their hunting by scavenging significantly more than females do [Fun98, 1334-1336].

Although females are highly cooperative when working together to care for the young cubs, cooperation varies by individual females during hunting. They cooperate if the prey is difficult to capture, but are less likely to cooperate if the prey is easy to capture [Hei95, 1260]. After one of the females has started a hunt, her companions may or may not join her. If the prey is difficult, such as a buffalo or a zebra, the companions are more likely to participate in the hunt [Pac97, 57]. This essentially means that each lioness makes a choice about her own risk relative to the potential gain during the hunt.

When hunting or protecting, some females in a pride may cooperate unconditionally while others may cooperate only when needed. Each strategy provides for the “long-term rewards of protecting essential companions and a stable territory” [Hei95, 1262]. Yet, some females gain a short-term benefit by lagging behind during territorial disputes. This suggests the cooperative females can have a “great variety of behavioral strategies” [Hei95, 1262] from which to choose.

An accepted model for species that capture prey such as lions makes four predictions. First, the tendency to be involved in capturing prey increases with the difficulty involved in the capture but levels off if no advantage is gained by adding another hunter. Second, more proficient hunters have a higher chance of participating than do hunters with less proficiency. Third, the success of group hunting improves asymptotically as group size increases, causing each additional animal to have a smaller contribution. And fourth, the genetic relationship between the members of the hunting group leads to improved cooperation. [Sch91, 698]

While the members of a hunting group may begin a hunt within 50 meters of each other, each individual lion can choose to participate or not. The nature of participation is highly varied as each individual may travel varying distance toward the prey before a hunt is terminated. The movement toward the prey may be unrelated to the hunt such as when trying to remain with companions or as an effort to be perceived as participating. The choice between being a conditional and unconditional hunter is a behavior that reveals itself “only under specific conditions” [Sch91, 699].

So, lions are a fairly good example of the nature of cooperation to be used in modeling the interaction of individual entities. They exhibit a range of choices in pursuit of a goal, namely the prey that will provide the pride’s next meal. There is a lead hunter that initiates the hunt for a specific prey and then cooperating companions make a choice as to how they will participate, either by joining in the pursuit or sitting by the sidelines.

CHAPTER 3
DESCRIPTION OF APPROACH

The approach used for this thesis involved a balance between attempting to achieve realism while showing that simple rules could be used to achieve cooperation among agents in pursuit of a goal. The most direct approach appeared to be the use of a predator-prey model where the predators could be given simple cooperative rules while pursuing a prey with simple rules for evasion. Further, using an object-oriented implementation provided a means of designing the program with a touch of realism. The artificial world became an individual object that is an instance of the Environment class. A Plant class could be used for all the operations performed by plants in the environment. Additionally, classes for predators and prey could be used to abstract the attributes and methods used by these "creature" agents.

A Predator-Prey Program

A simple predator-prey model was chosen as the program's basis. Although the model is loosely based on the cooperation of lions during a hunt, no specific types of animals were chosen for the modeling. Only the most basic concepts of predators and prey were designed into the program. But, this conceptual basis of predators and prey was associated with the concepts of agents and multi-agent systems. In particular, the predator-prey model was related directly to an artificial world containing goal-oriented agents capable of influencing their world as well as communicating among themselves.

The program uses an object-oriented design and is implemented using the Java programming language. As such, the artificial world and simple agents—the predators and prey—are created as objects when the program is executed. The cooperation among the agents is implemented as method calls between the objects.

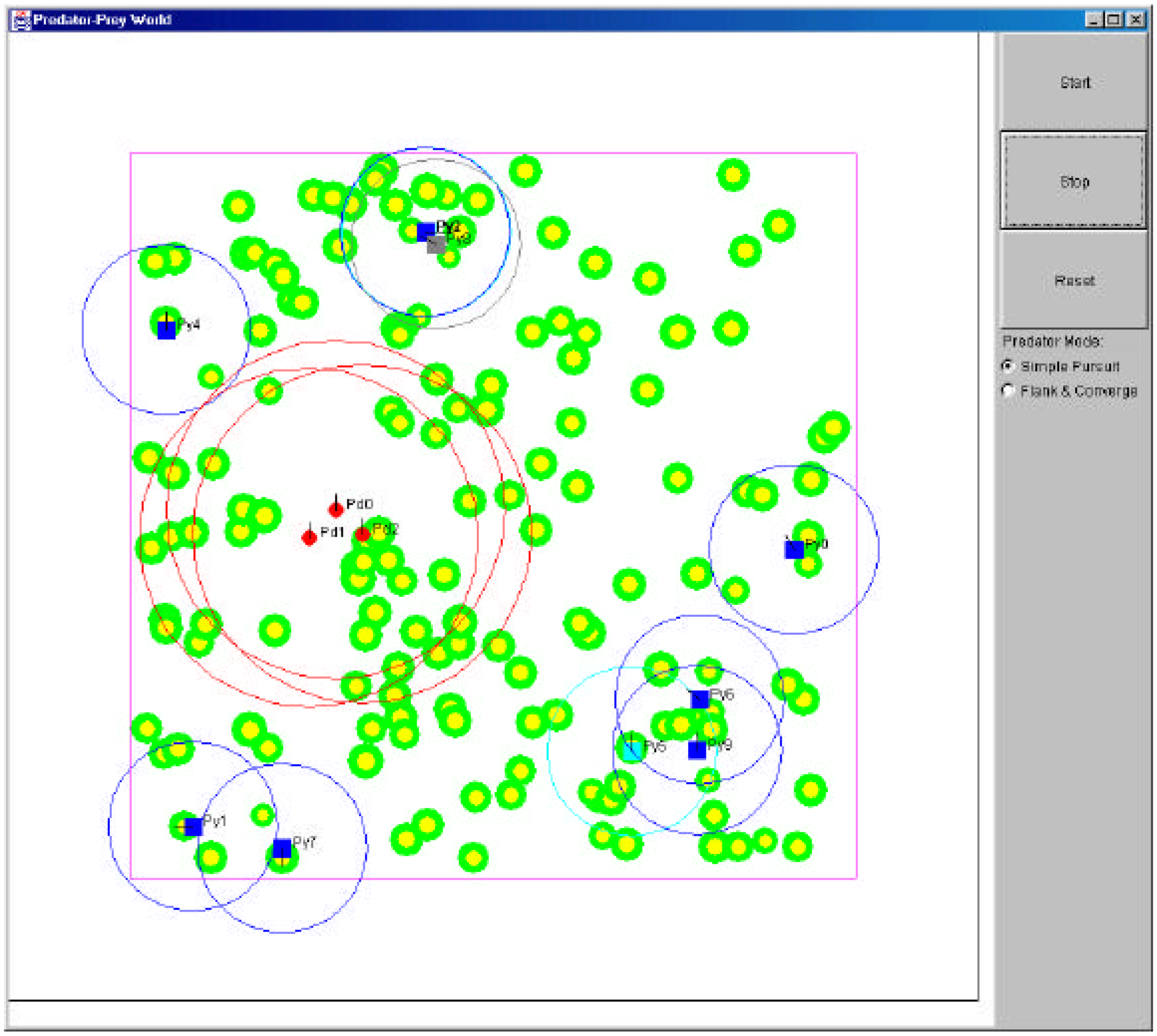

Figure 3-1. Predator-Prey Program's User Interface

The environment, the artificial world, of the program consists of a bounded zone with a size of 800 by 800 pixels. Plants and creatures exist within the environment with the creatures divided into groups of three predators or ten prey. As shown in Figure 3-1, the environment is divided into two zones. The inner zone, 600 by 600 pixels, is the area where the predators look for prey and the prey look for food. The 100-pixel wide outer zone surrounds the inner zone and serves as a safe haven for the prey to retreat from predators. This safe zone represents an area that the predators cannot enter. The plants are created in the inner zone of the environment. They have the capacity for growth and provide food for the prey to consume. One hundred fifty plants are created and randomly positioned when the environment is constructed.

The prey creatures are created at random locations within the inner zone of the environment. Their primary goal is moving from plant to plant and consuming food at each plant. They are given a limit range of vision for spotting both plants and predators. They can see any plants within this visual range but have only a percentage chance of spotting predators. When a predator is spotted, the prey attempt evasion using the simple rule: move in a direction mostly opposite to the predators unless the safe zone is within visual range, in which case, move a little more toward the safe zone in addition to moving opposite to the predators.

The predator creatures are created near the center of the inner zone and begin by wandering randomly. The movement of the predators is limited to the inner zone of the environment. The visual range of predators is approximately twice that of the prey as is their maximum speed when pursuing prey. As they wander, the three predators stay together in a roughly triangular pattern with a lead predator in front followed by the other two, one to the right side and the other to the left side of the lead. The lead predator is responsible for choosing the target prey, which is then communicated directly to the other two predators. Once a prey is targeted, the prey is stalked using one of two predator operation modes: a simple pursuit mode and a flank and converge mode.

The simple pursuit mode used by the predators begins as soon as the lead predator chooses a target prey. Each predator increases its movement to the maximum speed and begins pursuing the chosen prey. This is the simplest form of predator cooperation in the program. Other than allowing the lead predator to get to the target prey first, no direct interaction occurs between the predators after the pursuit begins. The simple rule being used is: each predator performs the same actions, pursuing the target at maximum speed until it is caught or it escapes.

The flank and converge mode is slightly more involved, requiring more cooperation among the predators. The lead predator still chooses the target prey and moves to a position just outside the visual range of the target. The other two predators move to positions on opposite sides of the target, essentially flanking the given prey. Once all three predators are in position, they converge on the prey at maximum speed. The simple rules used are: 1) If predator is lead, attempt to maintain a given distance from the target prey, 2) If predator is not the lead, move away from lead and toward a flanking position beside the target prey, while trying to stay outside the visual range of the prey, 3) If a non-lead predator is in position and the prey moves farther away, abandon the position and move to closer flanking position, and 4) If all three predators are in position, converge on target prey at maximum speed.

Under both predator modes, a prey is pursued until all three predators capture it or it escapes by entering the safe zone where the predators cannot go. To be considered captured, all three predators must occupy the same pixel as the target prey, at which time the target prey ceases to function and is considered “dead” in the environment. To escape, the target prey must enter the safe zone while being pursued, at which time it is considered to have “escaped” and no longer operates in the environment. Prey entering the safe zone while not being pursued are still considered alive and must continue to seek food. Once a prey reenters the inner zone, it is subject to pursuit by the predators.

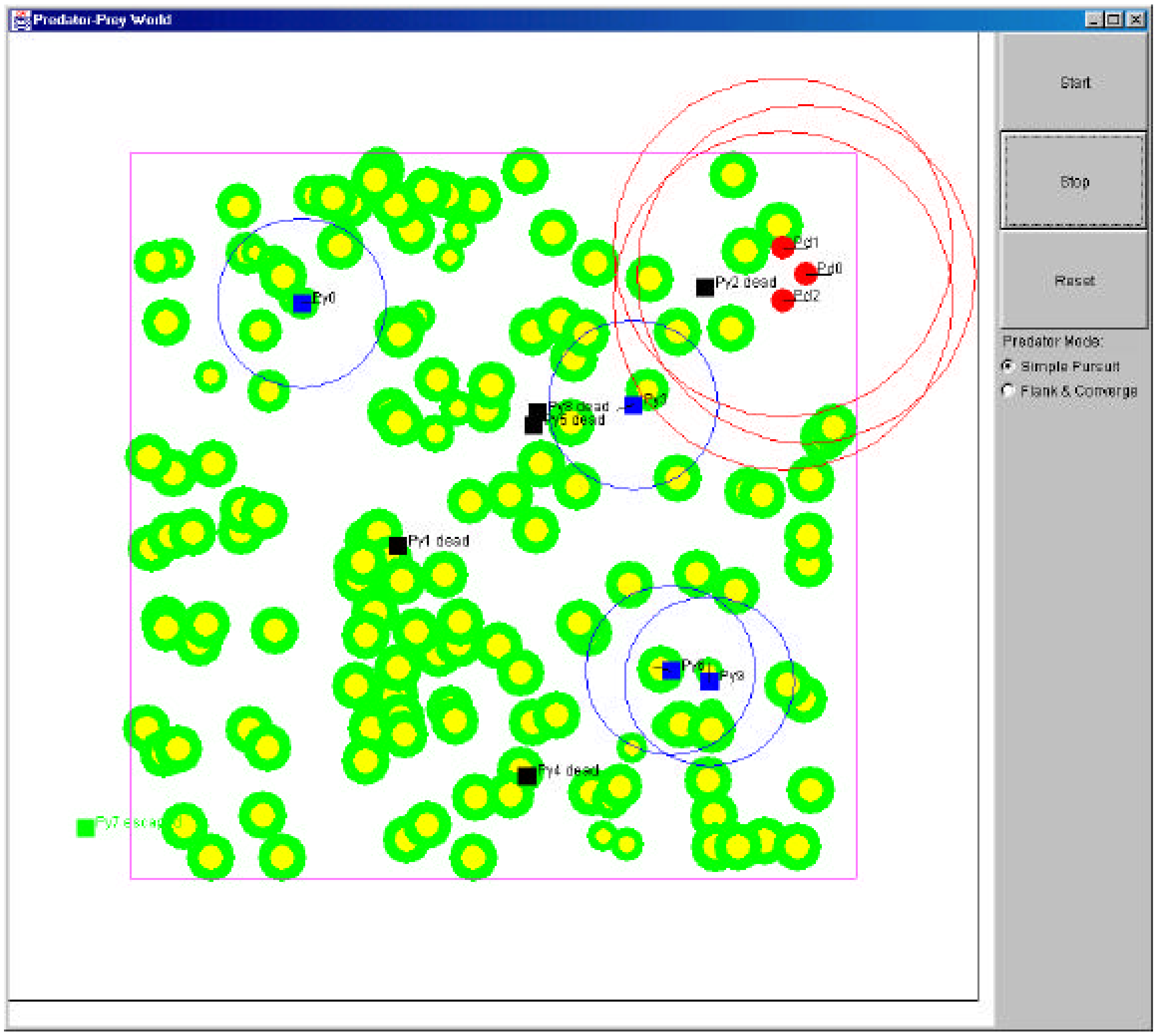

Figure 3-2. Sample Screen of Predator-Prey Program

Figure 3-2 shows an example screen image of an execution of the predator-prey program. The figure shows the formation of predators in the upper right corner of the environment, shortly after they captured prey number 2. In the lower left corner is the marker showing that prey number 7 escaped, having entered the safe zone while being pursued. The circles surrounding the creatures indicate the visual range of each creature.

The details of the predator-prey model described above were matched against an objected oriented design consisting of several classes created using the Java language. The remainder of this chapter is devoted to describing these classes with the actual code given in Appendix A. The description excludes the basic get and set methods used to access various attributes of each class since their operations are clear.

PredPrey Class

This PredPrey class serves as the main class for the program. It is responsible for creating the user interface and initiating the application.

PredPrey Constructor

The PredPrey object contains the primary frame or window where the program's activities occur. This constructor creates the new instance of the frame and handles the setting of various parameters for the frame.

Main Method

When the program starts, the user interface manager of the Java Swing package is used to create the windowing environment for the program. It then creates the instance of the PredPrey class.

PredPreyFrame Class

This class serves to define the frame where all the visual elements of the program are shown to the user. It also creates a timer used to execute turns of the predator-prey environment.

PredPreyFrame Constructor

The constructor enables windows based events, such as the timer and button clicks, as well as initializing the running of the frame via the jbInit method described next.

JbInit Method

This method is responsible for creating the visible components of the user interface and setting the connection between the various buttons and the appropriate program elements. It creates panels for the 800 x 800 pixel “world” object as well as the panel for push buttons and radio buttons. The push buttons are for starting, stopping, and resetting the world object. The radio buttons allow the user to choose between the two predator operation modes.

ActionPerformed Method

This method controls the program’s response to the clicking of the push buttons as well as timer events. When the start button is pressed, a timer is created and is started with an event signal created at 10 millisecond intervals. When the stop button is pressed, the timer is stopped. The reset button creates a new environment object, assigning it to the world variable, and then executes the first turn of the environment object. When a timer signal is received, the runWorld method is executed to perform a turn of the environment object.

ItemStateChanged Method

This method handles the response to changes in the radio buttons. If “simple pursuit” is selected, the predator mode is set to one, representing predators using the simple-pursuit operating mode. If the “flank & converge” choice is made, the predator mode is set to two for the flank and converge mode of operation.

RunWorld Method

This method tells the world object, an instance of the Environment class, to perform a turn and then repaints the window to show the results of the turn.

EnvironPanel Class

This class permits the overriding of the paintComponent method. The PaintComponent method is responsible for drawing the window as well as drawing the lines on the screen that represent the boundaries of the inner and outer zones of the artificial 800 by 800 pixel world.

Environment Class

This class is responsible for creating the artificial world where the plants and creatures exist. Its methods provide the sharing of information about its contents as well as its operations from turn to turn.

Environment Constructor

The constructor creates the artificial world with 150 plants, ten prey creatures, and three predator creatures, one lead with two subordinate predators.

GiveTargetPlants Method

This method creates and returns a linked list of the plants within a given distance of a point within the environment.

GivePredatorsInRange Method

This method creates and returns a linked list of the predators within a given distance of a point within the environment.

GivePreyInRange Method

This method creates and returns a linked list of the prey that have not escaped, are not captured, and are within a given distance of a point within the environment.

ProcessPlants Method

This method instructs each plant to perform its activities for the current turn.

ProcessCreatures Method

This method instructs each prey creature to perform its activities for the current turn. It then repeats this process for each predator.

PerformTurn Method

This method is responsible for performing the sequence of method calls within the environment for a given turn. First, it processes the plants and then the creatures within the environment. Lastly, the method increases the turn number.

DrawEnvirons Method

This method handles the drawing of the plants and creatures within the environment. It instructs each plant and each creature to draw itself.

AdjustCoord Method

This method takes as its input a point in the environment as well as a boundary parameter. If the given point is outside the bounded area, its x and y coordinates are adjusted to be the closest point within the bounded area.

AdjustLeadCoord Method

This method is essentially the same as the adjustCoord method except tailored to the lead predator's need to avoid the edge of the bounded area.

ComputeDist Method

This method computes the distance between two points within the environment by calculating the square root of the sum of the square of the difference in y coordinates and the square of the difference in x coordinates.

Plant Class

The plant class is responsible for the operations of the plants within the environment. Each plant has a specific location, a growth rate, and a set of size related parameters.

Plant Constructor

The constructor creates a plant with a random growth rate between 200 and 600 turns and a random location within the inner zone of the environment. Each plant is green with a yellow center. The minimum size of a plant is a circle with a 10-pixel radius and its maximum is a 20-pixel radius. A plant will grow in radius by one pixel in each turn that is a multiple of its growth rate.

PerformActivities Method

This method allows the plant to grow if it is an appropriate turn based on its growth rate.

YieldFood Method

This method decreases the size of the plant by a requested amount down to its minimum size.

Draw Method

This method draws the concentric yellow and green circles based on its location and size.

Creature Class

This class embodies the common elements of both prey and predators within the environment. It serves as the parent class for the two types of creatures. It includes the attributes for size, speed, visual range, current location, target location, and skin color. Also, values are kept for the turn when the creature can make its next move as well as a representation of the environment for making information requests.

Creature Constructor

The constructor sets only the most basic values of the creature’s ID number, the environment, and initial movement turn.

RandDist Method

This method computes a random number based on the visual range, which is a value between the negative range and positive range values.

Move Method

This method computes the next closest pixel in the direction of the target location. It also determines the end point for the direction line that is drawn from the creatures center point the direction of the target location.

Prey Class

This class is a child class of the creature class described above. It includes attributes for tracking recently visited plants, whether or not it has been captured or escaped, whether or not it is being pursued, and whether or not it has spotted a predator. Additionally, its visual range is divided into 10 increments used for computing percentage chances of spotting a predator at the given incremental distance.

Prey Constructor

The constructor creates a prey creature at a random location within the 600 by 600 pixel inner zone. It also divides the visible range of the prey into increments that are approximately the size of 0.09 times the visible range of the prey, starting at 20 percent of the visible range and continuing to the limit of the visible range. The constructor additionally sets the prey as not being pursued, not captured or escaped, and not having spotted a predator. The skin color of the prey is a random choice among cyan, blue, and gray.

InSafeZone Method

This method determines whether or not the prey is in the outer zone where x or y coordinates are less than or equal to 100, or greater than or equal to 700. If the prey is in the outer zone, the method returns a true value; otherwise, it returns a false value.

PerformActivities Method

This method controls the sequence of activities performed during a turn by the prey. If the prey has been captured or has escaped, the method does nothing. Each turn's activities begin with seeing if any predators are within range. If predators are within range and have not previously been spotted, an attempt to spot them is made via the wasPredatorSpotted method described below. If during any turn a predator is spotted, speed is increased to maximum speed and an attempt to evade the predator(s) is made. If a predator was not spotted, normal activities are performed.

WasPredatorSpotted Method

This method determines whether or not any of the predators in a given linked list are spotted. A prey's visible range is divided into 11 concentric increments. Within the innermost circle that extends a distance equal to 20 percent of the visible range, there is a 100 percent chance of spotting a predator. In each of the remaining increments, the percentage chance of being spotted decreases by 10 percent, leaving only a 10 percent chance of spotting any given predator at the limits of the prey's visible range.

EvadePredator Method

This method calculates the next most desirable location for the prey when it is attempting to evade one or more predators. It attempts to move in a direction approximately opposite to the predators. If the safe zone is within visual range, movement in the direction of the safe zone is given a higher weight in the computation.

PerformNormalActivities Method

This method covers the activities that occur when no predators are spotted. The prey attempts to travel from plant to plant, consuming food at each plant it encounters. It tries to avoid returning to recently visited plants.

Draw Method

This method determines how the prey is shown on screen. If the prey has escaped, it is drawn as a green square with the word “escaped” beside it. If the prey has been captured, it is drawn as a black square with word “dead” beside it. Otherwise, the prey is drawn as a square with the color defined by its skin color attribute.

Predator Class

This class is a sub-class of the Creature class described above. It defines the predators within the environment. Each predator has an operation mode based on the value chosen by the user (i.e., simple pursuit or flank and converge). It also can identify its target prey and its lead predator. The predator also tracks whether or not it is wandering, in pursuit of a prey, or in position to pursue. Additional attributes are used for reporting information to the user about prey pursuits such as the number of turns pursued and the distance traveled before escape or capture.

Predator Constructor

This constructor creates a predator within 25 pixels of the center of the inner zone of the environment. The skin color of the predator is a random choice among magenta, red, and dark gray. Each predator initially wanders without a target prey.

PerformActivities Method

This method controls the nature of the activities performed each turn by the predators. If it is turn number zero and the lead predator performing the action, a target location is chosen at random within visual range to prevent the initial pursuit of a prey that happens to be randomly located near the predators. If no prey is targeted and the predator is wandering, the performWandering method is invoked. If no prey is targeted and the predator is not wandering, the seekPrey method is performed. Otherwise, a prey has been targeted. If the target prey has been captured or has entered the safe zone, the predator returns to wandering at normal speed with the lead predator choosing a new random target location. If the predator is the lead predator and was in pursuit of the prey, pursuit data is printed. If the target prey has not yet been captured or entered the safe zone, the prey is either pursued via the pursuePrey method or a capture attempt is made using the prey's setCaptured method that requires all three predators to be in the same location as the prey.

PerformWandering Method

This method is used when a predator is wandering during the current turn. If the predator is the lead predator, a check is made to see if it has reached the target location. If it has, the lead predator attempts to seek a new prey target. If the predator is not the lead predator, it attempts to seek the prey target if the lead has found one. Otherwise, the predator moves relative to the lead predator's location.

MoveRelativeToLead Method

This method is used by the predators that are not the lead predator to determine where the next move should be. First, the direction the lead predator is moving is calculated. Next, positions are calculated that are 30 pixels behind the lead predator and to the left or right hand side of the lead depending upon which predator is moving. Predator one tends to stay to the left side of the lead while predator two tends to stay toward the right side.

SeekPrey Method

This method determines if a target prey is available within the visual range of the predators. If the predator is not the lead predator, the method checks to see if the lead predator has chosen a target prey. If the lead has, it becomes the target prey of the predator. Otherwise, the predator continues wandering. If invoked by the lead predator, this method checks to see if there are prey creatures within visual range and not in the safe zone. From the prey within range, the lead predator chooses the closest one as the new target prey. If there is no prey within range, the lead predator picks a random location and continues wandering.

PursuePrey Method

This method performs either the simplePursuit method or the flankAndConverge method depending on which operation mode was chosen for the predator.

SimplePursuit Method

This method is called when the predator is using the “simple pursuit” operation mode. It updates the target location for the predator based on the changing location of the target prey. Additionally, if the predator is not the lead predator, the method determines if the predator is closer to the target prey than the lead predator, adjusting the predator’s speed accordingly.

FlankAndConverge Method

This method is called when the predator is using the “flank and converge” operation mode. If, during a turn, the predator is in pursuit of the target prey, the predator’s speed is adjusted to allow the lead predator to arrive at the prey first. If the predator is in position and the other two are also in position, the pursuit of the prey can begin. If the predator is in position, not in pursuit, and the target prey moves too far away, the predator is no longer in position and will attempt to move closer to the target prey. When the predator is not in position and not in pursuit, the method calls the determineFlankingPosition method to coordinate the motion of the predators relative to each other and the target prey.

DetermineFlankingPosition Method

This method determines what should occur as the predators move into flanking position relative to the target prey. If the predator is the lead predator, the method checks to see if the distance to the prey is just beyond the target prey's visual range. If the distance is appropriate, the lead predator is considered to be in position and remains in position until the target prey is either pursued and captured or escapes. If the predator is not the lead predator, the method calls the moveRelativeToLead method if the lead predator is not yet positioned or calls the calcFlankLocation method to plot the movement of the predator into a flanking position.

CalcFlankLocation Method

This method calculated the next location for the predator as it moves toward a location that flanks the target prey. While the predator is still somewhat near the lead predator, the calculated location is almost directly left or right of the lead predator. Once the predator has moved sufficiently to either side of the lead predator, the method then calculates a target location that is to the left or right side of the target prey and plus or minus 50 degrees from the line drawn between the target prey and the lead predator.

Draw Method

This method draws the circle that represents the predator on screen. The color of the circle is based on the skin color randomly selected during predator construction.

Conclusion

Thus, the predator-prey program uses an object-oriented implementation of a simple rule-based model of predator cooperation. The program is divided in a collection of classes including the PredPrey and PredPreyFrame classes that provide the basic GUI operations, the Environment class that acts as the "artificial world," the Plant class that serves as the food supply for the prey, and the Predator and Prey classes (sub-classed from the Creature class) that represents the animals in the environment. In the next chapter, the results of executing the program are discussed together with a comparison of the two predator modes.

CHAPTER 4
RESULTS OF COMPUTATIONAL EXPERIMENTS

This chapter discusses the predator-prey program's execution results using the different predator modes. To facilitate the collection of data from one execution to the next, the actionPerformed method of the PredPreyFrame class was modified to limit the number of turns to 20,000, which was usually sufficient, but not always sufficient, to allow the predators to pursue the 10 prey creatures. Some runs of the program resulted in fewer than 10 pursuits. Also, each mode was used 20 times to obtain a sufficient number of prey pursuits to compare the efficiency and effectiveness of each mode. The comparison is based on the number of captures or escapes relative to the number of pursuits during the collection of 20 runs.

For each prey pursuit, data was collected related to the number of turns accumulated from the start of the pursuit until the prey either was captured by the three predators or escaped into the safe zone. Also, the distance traveled by the prey, measured in number of pixels, was computed based on the prey's position at the start of the pursuit and the prey's position at the end of the pursuit. The comparison of the two predator modes is discussed in the following paragraphs and is summarized in Tables 4-1 through 4-3. The raw data is provided in Appendix B.

During an execution of the predator-prey program, each prey is pursued no more than once with the pursuit ending when the prey is either captured or escapes in the safe zone. To be considered to have escaped, the prey must enter the safe zone while being pursued.

If a prey enters the safe zone at any other time, the act is treated as a “retreat” and the prey must return to the inner zone.

Table 4-1. Prey Pursuits by Predator Mode

Mode	No. of Pursuits	No. of Captures	No. of Escapes
Simple Pursuit	198	153	45
Flank and Converge	192	160	32

As indicated in Table 4-1, because of the randomness used in the “artificial world,” the two modes resulted in different numbers of prey pursuits during the 20 executions of each mode. The simple pursuit mode yielded 198 pursuits of which 153 ended in captures and 45 ended in escapes. The flank and converge mode yielded 192 pursuits of which 160 resulted in captures and 32 yielded escapes. The percentage of escapes was 22.7 percent for the simple pursuit mode and 16.7 percent for the flank and converge mode. The flank and converge mode showed approximately a 26 percent improvement over the simple pursuit mode in number of escapes.

Table 4-2. Prey Escape Data Per Predator Mode

	Simple Pursuit	Flank and Converge
Average No. Of Turns	97 turns	75 turns
Minimum No. Of Turns	6 turns	13 turns
Maximum No. Of Turns	221 turns	153 turns
Average Distance Traveled	49 pixels	41 pixels
Minimum Distance Traveled	1 pixel	2 pixels
Maximum Distance Traveled	108 pixels	82 pixels

When pursuing a prey near the safe zone where the prey has the opportunity to escape, neither mode was particularly efficient, but the flank and converge mode showed an improvement over the simple pursuit mode. The simple pursuit mode used an average of 97 turns before the prey escaped and allowed the prey to travel an average distance of 49 pixels while the flank and converge mode used an average of 75 turns for pursuit and the prey traveled an average of 41 pixels. The flank and converge mode was more efficient than the simple pursuit mode by 23 percent in number of turns and by 16 percent in distance of pursuit. Yet, the biggest difference is apparent in the comparison of the maximum number of turns used to pursue a given prey, 221 turns for the simple pursuit mode and 153 turns for the flank and converge. A prey has a much better chance of escaping from the predators under the simple pursuit mode than under the flank and converge mode.

Table 4-3. Prey Capture Data Per Predator Mode

	Simple Pursuit	**Flank and Converge**
Average No. Of Turns	194 turns	122 turns
Minimum No. Of Turns	139 turns	78 turns
Maximum No. Of Turns	271 turns	161 turns
Average Distance Traveled	92 pixels	58 pixels
Minimum Distance Traveled	58 pixels	3 pixels
Maximum Distance Traveled	119 pixels	94 pixels

The most dramatic difference appears during the comparison of the two modes for captured prey. The simple pursuit mode required an average of 194 turns for pursuing the prey and allowed the targeted prey to travel an average distance of 92 pixels before capture. The flank and converge mode needed an average of 122 turns for pursuit and permitted the prey to travel 58 pixels on average. This represents a 37 percent improvement of the flank and converge mode over the simple pursuit mode. Also, significant differences appear when comparing of the minimum and maximum number of turns used as well as the minimum and maximum distances traveled by the prey while being pursued. The minimum number of turns was 139 for the simple pursuit mode and 78 for the flank and converge mode. The maximum number of turns was 271 for the simple pursuit mode and 161 for the flank and converge mode. The comparison indicates a 44 percent improvement of the flank and converge mode over the simple pursuit mode in minimum turns and a 40 percent improvement in maximum turns. In addition, the minimum distance traveled by a prey under the simple pursuit mode is 58 pixels, the same as the average distance under the flank and converge mode. So, the flank and converge mode is better able to capture prey, primarily by limiting the distance that the prey can travel once it is being pursued.

In summary, the simple rules of the flank and converge mode appear to be a better technique than the rules used by the simple pursuit mode. Improvements are as great as 44 percent for the flank and converge mode with an average advantage of 37 percent when pursuing a prey that is not near its safe haven. Even when the prey is near the safe zone, the flank and converge mode is better by 23 percent. In the next chapter, conclusions are discussed together with thoughts about future avenues of research based on these results.

CHAPTER 5
CONCLUSIONS

Multi-agent systems allow for problems to be solved that would be difficult to solve by a single agent. The basics of the predator-prey model appear to be a good example of a situation where several “agent” predators working together can solve a problem that would be difficult to solve alone. In particular, the cooperative hunting of prey by a group of predators provides a basis for a simple comparison of methods of cooperation.

From the results of executing the predator-prey program, enough data was available to compare two distinct methods of predator cooperation. Although the program was based on a simple model, the visual element of the program’s execution gave the impression of actual pursuits occurring. In a sense, the predators pursued their goal—the prey—with deliberate intention. By the addition of simple rules to govern their cooperation in the flank and converge mode, the predators achieved their goal more readily. Through the use of a little information sharing about positioning for pursuit, a dramatic increase in efficiency and effectiveness was achieved.

A future avenue of research based on the use of simple rules for cooperation is more direct representation of various wildlife groups. By incorporating additional attributes and methods in the Predator class based on animal characteristics observed by wildlife biologist, it should be possible to develop an accurate computational model of predators such as lions in various environments. Similar changes could be made in the Prey class to allow the prey to have more complex rules for evasion or even have an ability to fight back against a predator. Also, even though the predators in the predator-prey program were faster during the entire prey pursuit, this is not always the case in the wild. Changes could be made to more realistically portray the actual speed capabilities of predators and prey.

Separate characteristics for male and female genders in both prey and predator could be added. This would allow for possible comparisons of nurturing techniques for females and of territorial protection techniques by males.

Through the addition of a more accurate portrayal of food consumption, the prospects of survivability could be modeled using simple cooperative rules. Benefits of scavenging by a predator could be included. Also, battles between competing groups might be introduced into the program. Larger groups of predators and prey of varying size would be possible as well.

Another area for possible research is in simple modeling of military operations. Combat troops can be represented as predators while their target can serve as the prey. When troops move, they usually stay in close formation based on simple rules. When engaging the enemy, operations similar to flank and converge are also used.

Thus, simple rules based on predator-prey modeling can be used in a multi-agent system. By using simple rules, a direct comparison of efficiency or effectives of adding a rule can be made. Further, avenues of future research exist for extending the use of predator-prey modeling.

APPENDIX A
CODE FOR THE PREDATOR-PREY PROGRAM

This appendix contains the Java-based implementation of the object-oriented classes described in Chapter 3. The program includes seven files: PredPrey.java, PredPreyFrame.java, Environment.java, Plant.java, Creature.java, Prey.java, and Predator.java.

Predprey.java

```
import javax.swing.UIManager;
import java.awt.*;

public class PredPrey {

boolean packFrame = false;

/**Construct the application*/
public PredPrey() {
  PredPreyFrame frame = new PredPreyFrame();
  //Validate frames that have preset sizes
  //Pack frames that have useful preferred size info
  if (packFrame) {
    frame.pack();
  }
  else {
    frame.validate();
  }
  //Center the window
  Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize = frame.getSize();
  if (frameSize.height > screenSize.height) {
    frameSize.height = screenSize.height;
  }
  if (frameSize.width > screenSize.width) {
    frameSize.width = screenSize.width;
  }
  frame.setLocation((screenSize.width - frameSize.width) / 2,
                    (screenSize.height - frameSize.height) / 2);
  frame.setVisible(true);
} // end PredPrey()

/**Main method*/
public static void main(String[] args) {
  try {
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
  }
  catch(Exception e) {
    e.printStackTrace();
  }
  new PredPrey();
} // end main()
} // end class PredPrey
```

PredPreyFrame.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PredPreyFrame extends JFrame
       implements ActionListener, ItemListener {

JPanel contentPane;
BorderLayout borderLayout1 = new BorderLayout(5, 5);
JButton startButton = new JButton("Start");
JButton stopButton = new JButton("Stop");
JButton resetWorldButton = new JButton("Reset");
JPanel displayPanel = new EnvironPanel(); // class is defined below
Timer timer;
public static final int ENVIRON_BOUNDARY = 800;
static Environment world = new Environment(ENVIRON_BOUNDARY, 1);
int selectedPredatorMode = 1; // default to first mode
JRadioButton jrb_pred1, jrb_pred2; // radio buttons for mode

/**Construct the frame*/
public PredPreyFrame() {
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

/**Component initialization*/
private void jbInit() throws Exception  {
  contentPane = (JPanel) this.getContentPane();
  contentPane.setLayout(borderLayout1);
  this.setSize(new Dimension(950, 850));
  this.setTitle("Predator-Prey World");
  // create container for panels
  Container container = getContentPane();
  // create panel for buttons
  JPanel controlPanel = new JPanel();
  controlPanel.setLayout(new GridLayout(10, 1));
  // add buttons
  controlPanel.add(startButton);
  controlPanel.add(stopButton);
  controlPanel.add(resetWorldButton);
  // add listeners for events
  startButton.addActionListener(this);
  stopButton.addActionListener(this);
  resetWorldButton.addActionListener(this);
  // create and add radio buttons for two predator modes
  JPanel predRadioButtons = new JPanel();
  ButtonGroup predRadioGrp = new ButtonGroup();
  predRadioButtons.setLayout(new GridLayout(4, 1));
  predRadioButtons.add(new Label("Predator Mode:"));
  predRadioButtons.add(jrb_pred1 = new JRadioButton("Simple Pursuit",
true));
  predRadioButtons.add(jrb_pred2 = new JRadioButton("Flank & Converge",
false));
  predRadioGrp.add(jrb_pred1);
  predRadioGrp.add(jrb_pred2);
  jrb_pred1.addItemListener(this);
  jrb_pred2.addItemListener(this);
  controlPanel.add(predRadioButtons);
  // add control panel in east side of container
  container.add(controlPanel, BorderLayout.EAST);
  // create panel to display world
  displayPanel.setBackground(Color.white);
  // add to container frame
  container.add(displayPanel, BorderLayout.CENTER);
}

/**Overridden so program exits when window is closed*/
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    System.exit(0);
  }
}

public static Environment getWorld() {
  return world;
} // end getWorld()

public void actionPerformed (ActionEvent e) {
  if (e.getSource() == startButton) {
    timer = new Timer(10, this);
    timer.start();
  }
  else if (e.getSource() == stopButton) {
    if (timer != null)
       timer.stop();
  }
  else if (e.getSource() == resetWorldButton) {
    world = new Environment(ENVIRON_BOUNDARY, selectedPredatorMode);
    runWorld(); // executes the turn number zero
  }
  else if (e.getSource() == timer) {
    runWorld();
  }
```

```
} // end actionPerformed()

public void itemStateChanged(ItemEvent e) {
  if (jrb_pred1.isSelected())
    selectedPredatorMode = 1;
  if (jrb_pred2.isSelected())
    selectedPredatorMode = 2;
} // end itemStateChanged

public void runWorld() {
  world.performTurn();
  repaint();
} // end runWorld()
} // end class PredPreyFrame


class EnvironPanel extends JPanel {

public void paintComponent(Graphics g) {
  super.paintComponent(g);
  PredPreyFrame.getWorld().drawEnvirons(g);
  // draw environment boundry lines
  g.setColor(Color.black);
  g.drawLine(0, 801, 801, 801);
  g.drawLine(801, 0, 801, 801);
  // draw plant area boundry lines
  g.setColor(Color.magenta);
  g.drawLine(100, 100, 700, 100);
  g.drawLine(100, 100, 100, 700);
  g.drawLine(100, 700, 700, 700);
  g.drawLine(700, 100, 700, 700);
} // end paintComponent()
} // end class EnvironPanel
```

Environment.java

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

public class Environment {

int turnNumber; // tracks number of turns performed
static public int environBounds;
static int predatorMode;
LinkedList preyList = new LinkedList();
LinkedList predList = new LinkedList();
LinkedList plantList = new LinkedList();

public Environment(int bounds, int predMode) {
  environBounds = bounds;
  predatorMode = predMode;
  int i, j;
  Plant currPlant;
  Creature currAnimal;
  turnNumber = 0;
```

```
    // add plants
    for (i = 0; i < 150; i++) {
      currPlant = new Plant(this);
      plantList.add(currPlant);
    }
    // add "prey" animals
    for (i = 0; i < 10; i++) {
      currAnimal = new Prey(this, i);
      preyList.add(currAnimal);
    }
    // add "predator" animals
    Predator leadPred = new Predator(this, 0, null, predatorMode);
    predList.add(leadPred);
    for (i = 1; i < 3; i++) {
      currAnimal = new Predator(this, i, leadPred, predatorMode);
      predList.add(currAnimal);
    }
    if (predatorMode == 1)
      System.out.println("Mode: Simple Pursuit");
    else
      System.out.println("Mode: Flank & Converge");
  } // end Environment()

  public int getTurnNumber() {
    return turnNumber;
  }

  public LinkedList giveTargetPlants(Point pt, int range) {
    LinkedList retList = new LinkedList();
    ListIterator li = plantList.listIterator();
    Plant currPlant;
    double dist;
    while (li.hasNext()) {
      currPlant = (Plant) li.next();
      dist = computeDist(pt, currPlant.getLocation());
      if (dist <= range) {
        retList.add(currPlant);
      }
    }
    return retList;
  } // end giveTargetPlants()

  public LinkedList givePredatorsInRange(Point pt, int range) {
    LinkedList retList = new LinkedList();
    ListIterator li = predList.listIterator();
    Predator currPred;
    double dist;
    while (li.hasNext()) {
      currPred = (Predator) li.next();
      dist = computeDist(pt, currPred.getLocation());
      if (dist <= range) {
        retList.add(currPred);
      }
    }
    return retList;
  } // end givePredatorsInRange()
```

```
public LinkedList givePreyInRange(Point pt, int range) {
  LinkedList retList = new LinkedList();
  ListIterator li = preyList.listIterator();
  Prey currPrey;
  double dist;
  while (li.hasNext()) {
    currPrey = (Prey) li.next();
    if (!(currPrey.isCaptured() || currPrey.inSafeZone())) {
      dist = computeDist(pt, currPrey.getLocation());
      if (dist <= range) {
        retList.add(currPrey);
      }
    }
  } // end while(...)
  return retList;
} // end givePreyInRange()

public void ProcessPlants() {
  ListIterator li = plantList.listIterator();
  Plant currPlant;
  while (li.hasNext()) {
    currPlant = (Plant) li.next();
    currPlant.performActivities(turnNumber);
  }
} // end ProcessPlants()

public void ProcessCreatures() {
  Creature currAnimal;
  // prey list
  ListIterator li = preyList.listIterator();
  while (li.hasNext()) {
    currAnimal = (Creature) li.next();
    currAnimal.performActivities(turnNumber);
  }
  // predator list
  li = predList.listIterator();
  while (li.hasNext()) {
    currAnimal = (Creature) li.next();
    currAnimal.performActivities(turnNumber);
  }
} // end ProcessCreatures()

public void performTurn() {
  // plants perform actions first
  ProcessPlants();
  // creatures perform actions second
  ProcessCreatures();
  // set for next turn
  turnNumber++;
} // end performTurn()

public void drawEnvirons(Graphics g){
  ListIterator li;
  Creature curranimal;
  Plant currplant;
  // plants
  li = plantList.listIterator();
  while (li.hasNext()) {
    currplant = (Plant) li.next();
    currplant.draw(g);
  }
  // prey
  li = preyList.listIterator();
  while (li.hasNext()) {
    curranimal = (Creature) li.next();
    curranimal.draw(g);
  }
  // predators
  li = predList.listIterator();
  while (li.hasNext()) {
    curranimal = (Creature) li.next();
    curranimal.draw(g);
  }
} // end drawEnvirons()

public void adjustCoord(Point pt, int boundarySize) {
  int lowBoundary = 0 + boundarySize;
  int highBoundary = environBounds - boundarySize;
  if (pt.x < lowBoundary)
    pt.x = lowBoundary + 1;
  if (pt.x > highBoundary)
    pt.x = highBoundary - 1;
  if (pt.y < lowBoundary)
    pt.y = lowBoundary + 1;
  if (pt.y > highBoundary)
    pt.y = highBoundary - 1;
} // end adjustCoord(Point, int)

public void adjustLeadCoord(Point pt, int boundarySize) {
  int lowBoundary = 0 + boundarySize;
  int highBoundary = environBounds - boundarySize;
  if (pt.x < lowBoundary)
    pt.x = lowBoundary + (lowBoundary - pt.x);
  if (pt.x > highBoundary)
    pt.x = highBoundary - (pt.x - highBoundary);
  if (pt.y < lowBoundary)
    pt.y = lowBoundary + (lowBoundary - pt.y);
  if (pt.y > highBoundary)
    pt.y = highBoundary - (pt.y - highBoundary);
} // end adjustLeadCoord(Point, int)

public double computeDist(Point p1, Point p2) {
  double distx = (double) p1.x - p2.x;
  double disty = (double) p1.y - p2.y;
  double dist = Math.sqrt(distx*distx + disty*disty);
  return dist;
} // end computeDist()
} // end class Environment
```

Plant.java

```
import java.awt.*;
import java.math.*;

public class Plant {

int size;
int minSize;
int maxSize;
int growthRate;
Point Location;
Environment environ;
Color centerColor;
Color outerColor;

public Plant(Environment e) {
  environ = e;
  size = 12;
  minSize = 10;
  maxSize = 20;
  growthRate = (int)(Math.random() * 400) + 200;
  // place plant in bounded section of Environment
  Location = new Point((int)(Math.random()*(e.environBounds-226)+113),
                       (int)(Math.random()*(e.environBounds-226)+113));
  centerColor = Color.yellow;
  outerColor = Color.green;
} // end Plant()

public Point getLocation() {
  return new Point(Location);
} // end getLocation()

public void performActivities (int turnNum) {
  // grow in size
  if ((size != maxSize) && ((turnNum % growthRate) == 0))
    size++;
} // end performActivities()

public int yieldFood(int request) {
  int avail = size - minSize;
  if (avail >= request) {
    size = size - request;
    return request;
  }
  else {
    size = size - avail;
    return avail;
  }
} // end yieldFood()

public void draw(Graphics g) {
  g.setColor(outerColor);
  g.fillOval(Location.x - size, Location.y - size, size*2, size*2);
  g.setColor(centerColor);
  g.fillOval(Location.x - size/2, Location.y - size/2, size, size);
} // end draw()
} // end class Plant
```

Creature.java

```
import java.awt.*;
import java.math.*;
import java.util.*;

public class Creature {

int size;
int maxSize;
int speed;
int normalSpeed;
int maxSpeed;
int visionRange;
int creatureID;
int moveTurn; // next Turn in which creature can move
Point CurrLocation;
Point TargetLocation; // location of target within 800x800
Point dirLineEndPt; // end point of direction line
Environment environ;
Color skinColor;

public Creature(Environment e, int id) {
  environ = e;
  creatureID = id;
  moveTurn = 1;
} // end Creature()

public Point getLocation() {
  return new Point(CurrLocation);
} // end getLocation()

public Point getDirPoint() {
  return new Point(dirLineEndPt);
} // end getDirPoint()

public int getCreatureID() {
  return creatureID;
}

// default activities
public void performActivities(int turnNum) {
  move();
  moveTurn = turnNum + speed;
}

// computes a random distance to a spot within vision range
public int randDist() {
  return (int)(Math.random()*visionRange*2) - visionRange;
}

protected void move() {
```

```
double dist, distx, disty;
double x1, x2, y1, y2;
double slope = 0;
boolean vertLine = false, horiLine = false;
// initial values of x and y coordinates
x1 = (double) CurrLocation.x;
y1 = (double) CurrLocation.y;
x2 = (double) TargetLocation.x;
y2 = (double) TargetLocation.y;
// determine slope
if (x1 == x2) {
  vertLine = true;
}
else {
  slope = (y2 - y1) / (x2 - x1);
}
// compute new current location
if (vertLine) { // change in y coordinate only
  if (y2 > y1)
     CurrLocation.y = CurrLocation.y + 1;
  else if (y2 < y1)
     CurrLocation.y = CurrLocation.y - 1;
}
else {
  // determine new x coordinate
  if (x2 > x1)
     CurrLocation.x = CurrLocation.x + 1;
  else if (x2 < x1)
     CurrLocation.x = CurrLocation.x - 1;
  // determine new y coordinate
  if (y2 > y1)
    CurrLocation.y = CurrLocation.y + 1;
  else if (y2 < y1)
    CurrLocation.y = CurrLocation.y - 1;
}
// compute direction line endpoint
double theta;
int delta_x, delta_y;
if (vertLine) {
  dirLineEndPt.x = CurrLocation.x;
  if (y2 > y1)
     dirLineEndPt.y = CurrLocation.y + size;
  else
     dirLineEndPt.y = CurrLocation.y - size;
}
else if (y1 == y2) {
  if (x2 > x1)
    dirLineEndPt.x = CurrLocation.x + size;
  else if (x2 < x1)
    dirLineEndPt.x = CurrLocation.x - size;
  else
    dirLineEndPt.x = CurrLocation.x;
  dirLineEndPt.y = CurrLocation.y;
}
else{
  theta = Math.atan(slope);
  delta_x = (int)(Math.cos(theta) * size);
```

```
    delta_y = (int)(Math.sin(theta) * size);
    if (x2 < x1) {delta_x *= -1; delta_y *= -1;}
    dirLineEndPt.x = CurrLocation.x + delta_x;
    dirLineEndPt.y = CurrLocation.y + delta_y;
  }
} // end move()

// default draw function, overridden in subclasses
public void draw(Graphics g) {
  g.setColor(skinColor);
  g.drawRect(CurrLocation.x - (size/2), CurrLocation.y - (size/2),
            size, size);
} // end draw()
} // end class Creature
```

Prey.java

```
import java.awt.*;
import java.math.*;
import java.util.*;

public class Prey extends Creature {

Plant TargetPlant;
Plant[] RecentPlants;
final int SZ_RecentPlantsArray = 10;
boolean escaped;
boolean captured;
boolean beingPursued;
boolean spottedPred;
int[] RangeIncArray; // incremental ranges within 'visual' range
LinkedList targetPlantList; // list of plants within vision range
LinkedList predatorsInRange; // list of predators within vision range

public Prey(Environment e, int id) {
  super(e, id);
  CurrLocation =
      new Point((int)(Math.random()*(e.environBounds-199)+ 100),
                (int)(Math.random()*(e.environBounds-199)+ 100));
  TargetLocation = new Point(CurrLocation);
  dirLineEndPt = new Point(CurrLocation);
  size = 15;
  maxSize = 20;
  visionRange = 70;
  RangeIncArray = new int[10];
  int i; double j;
  for (i=0, j=0.2; i<9; i++, j+=0.09) {
    RangeIncArray[i] = (int) (j * visionRange);
  }
  RangeIncArray[9] = visionRange;
  switch ((int)(Math.random() * 3))
  {
  case 0: skinColor = Color.cyan;
          break;
  case 1: skinColor = Color.blue;
```

```
          break;
  case 2: skinColor = Color.gray;
          break;
  }
  TargetPlant = null;
  RecentPlants = new Plant[SZ_RecentPlantsArray];
  for (i=0; i<SZ_RecentPlantsArray; i++){
    RecentPlants[i] = null;
  }
  escaped = false;
  captured = false;
  beingPursued = false;
  spottedPred = false;
  speed = 4; // lower is faster
  normalSpeed = 4;
  maxSpeed = 2;
} // end Prey()

void clearRecentPlantArray() {
  for (int i=0; i<SZ_RecentPlantsArray; i++){
    RecentPlants[i] = null;
  }
}

public boolean isCaptured() {
  return captured;
}

public void setBeingPursued() {
  if (!beingPursued) {
    beingPursued = true;
  }
}

public boolean inSafeZone() {
  boolean retValue = false;
  if ((CurrLocation.x <= 100) || (CurrLocation.x >= 700) ||
      (CurrLocation.y <= 100) || (CurrLocation.y >= 700)) {
    retValue = true;
  }
  return retValue;
} // end inSafeZone()

public void setCaptured() {
  int numPred = 0;
  ListIterator li = predatorsInRange.listIterator();
  Predator currPred;
  while (li.hasNext()) {
    currPred = (Predator) li.next();
    if (CurrLocation.equals(currPred.getLocation())) {
      numPred += 1;
    }
  }
  if (numPred == 3)
    captured = true;
  else
    captured = false;
```

```
} // end setCaptured

public void performActivities(int turnNum) {
  if (captured || escaped)
    return;
  if (turnNum == moveTurn) {
    predatorsInRange =
        environ.givePredatorsInRange(CurrLocation, visionRange);
    ListIterator li = predatorsInRange.listIterator();
    if (predatorsInRange.size() > 0 && !spottedPred) {
      spottedPred = wasPredatorSpotted(li);
      if (spottedPred) {
        speed = maxSpeed;
      }
      else {
        speed = normalSpeed;
      }
    }
    else if (predatorsInRange.size() == 0) {
      spottedPred = false;
      speed = normalSpeed;
    }
    if (spottedPred) {
      evadePredator(turnNum);
    }
    else {
      performNormalActivities(turnNum);
    } // end if-else (spottedPred)
  } // end if (turnNum == moveTurn)
} // end performActivities()

boolean wasPredatorSpotted(ListIterator li) {
  double dist;
  Predator currPred;
  boolean spotted = false;
  boolean found = false;
  int chanceOfBeingSpotted;
  int i;
  int randNum;
  while (li.hasNext() && !spotted) {
    currPred = (Predator) li.next();
    dist = environ.computeDist(CurrLocation, currPred.getLocation());
    for (i=0; i<9 && !found; i++) {
      if (dist <= RangeIncArray[i]) {
        found = true;
      }
    }
    chanceOfBeingSpotted = 100 - (i * 10);
    if (chanceOfBeingSpotted == 100) {
      spotted = true;
    }
    else {
      randNum = (int) (Math.random() * 100);
      if (randNum < chanceOfBeingSpotted) {
        spotted = true;
      }
    }
  } // end while(li.hasNext()...)
  return spotted;
} //end wasPredatorSpotted()

void evadePredator(int turnNum) {
  int xChange = 0, yChange = 0;
  ListIterator li = predatorsInRange.listIterator();
  TargetPlant = null;
  Predator currPred;
  Point predLoc;
  while (li.hasNext()) {
    currPred = (Predator) li.next();
    predLoc = currPred.getLocation();
    if (predLoc.x < CurrLocation.x) {
      xChange += 1;
    }
    else if (predLoc.x > CurrLocation.x) {
      xChange -= 1;
    }
    if (predLoc.y < CurrLocation.y) {
      yChange += 1;
    }
    else if (predLoc.y > CurrLocation.y) {
      yChange -= 1;
    }
  } // end while (li.hasNext())
  // see if safe area is near and move toward it
  int range = (int)(0.5 * visionRange);
  if ((CurrLocation.x > 100) && (CurrLocation.x - 100 <= range)) {
    xChange -= 3;
  }
  if ((CurrLocation.y > 100) && (CurrLocation.y - 100 <= range)) {
    yChange -= 3;
  }
  if ((CurrLocation.x < 700) && (700 - CurrLocation.x <= range)) {
    xChange += 3;
  }
  if ((CurrLocation.y < 700) && (700 - CurrLocation.y <= range)) {
    yChange += 3;
  }
  // determine random movement at very least
  if (xChange == 0) {
    xChange = (int)Math.round(Math.random()*2-1);
  }
  if (yChange == 0) {
    yChange = (int)Math.round(Math.random()*2-1);
  }
  // compute new location
  TargetLocation.x = CurrLocation.x + xChange;
  TargetLocation.y = CurrLocation.y + yChange;
  environ.adjustCoord(TargetLocation, 0);
  move();
  moveTurn = turnNum + speed;
} //end evadePredator()

void performNormalActivities(int turnNum) {
  // grow in size
  if ((size != maxSize) && ((turnNum % 500) == 0)) {
    size++;
  }
  // perform activities at target
  if (CurrLocation.equals(TargetLocation)) {
    // check to see if recently pursued, but escaped
    if (inSafeZone() && beingPursued) {
      escaped = true;
      return;
    }
    // check for plant
    if (TargetPlant != null) {
      TargetPlant.yieldFood(2); // eat
      // add plant to array of recent plants
      for (int i=SZ_RecentPlantsArray-1; i>0; i--) {
        RecentPlants[i] = RecentPlants[i-1];
      }
      RecentPlants[0] = TargetPlant;
    }
    // determine new target
    targetPlantList =
        environ.giveTargetPlants(CurrLocation, visionRange);
    ListIterator li = targetPlantList.listIterator();
    Plant currPlant;
    int randDist;
    if (targetPlantList.size() > 1) {
      TargetPlant = null;
      while (li.hasNext() && TargetPlant == null) {
        currPlant = (Plant) li.next();
        if (!inRecentPlants(currPlant)) {
          TargetPlant = currPlant;
        }
      }
      if (TargetPlant == null) {
        TargetLocation.x = CurrLocation.x + randDist();
        TargetLocation.y = CurrLocation.y + randDist();
        environ.adjustCoord(TargetLocation, 100);
      }
      else {
        TargetLocation = TargetPlant.getLocation();
      }
    }
    else { // no plants within vision range
      TargetPlant = null;
      clearRecentPlantArray(); // allows same plants to be visited
      TargetLocation.x = CurrLocation.x + randDist();
      TargetLocation.y = CurrLocation.y + randDist();
      environ.adjustCoord(TargetLocation, 100);
    }
  }
  move();
  moveTurn = turnNum + speed;
} // end performNormalActivities(int turnNum)

public boolean inRecentPlants(Plant currPlant) {
  boolean inList = false;
  for (int i=0; i<SZ_RecentPlantsArray; i++) {
    if (RecentPlants[i] == currPlant) {
      inList = true;
    }
  }
  return inList;
} // end inRecentPlants(Plant)

public void draw(Graphics g) {
  if (captured) {
    g.setColor(Color.black);
    g.fillRect(CurrLocation.x - (size/2), CurrLocation.y - (size/2),
              size, size);
    g.drawString("   Py" + creatureID + " dead",
              CurrLocation.x, CurrLocation.y);
  }
  else if (escaped) {
    g.setColor(Color.green);
    g.fillRect(CurrLocation.x - (size/2), CurrLocation.y - (size/2),
              size, size);
    g.drawString("   Py" + creatureID + " escaped",
              CurrLocation.x, CurrLocation.y);
  }
  else {
    g.setColor(skinColor);
    g.fillRect(CurrLocation.x - (size/2), CurrLocation.y - (size/2),
              size, size);
    g.setColor(Color.black);
    g.drawLine(CurrLocation.x, CurrLocation.y,
              dirLineEndPt.x, dirLineEndPt.y);
    g.drawString("   Py" + creatureID, CurrLocation.x, CurrLocation.y);
    g.setColor(skinColor);
    g.drawOval(CurrLocation.x - visionRange,
               CurrLocation.y - visionRange,
               visionRange*2, visionRange*2);
  }
} // end draw()
} // end Class Prey
```

Predator.java

```
import java.awt.*;
import java.math.*;
import java.util.*;
import java.lang.Math;

public class Predator extends Creature {

Prey TargetPrey;
Predator leadPred;
int opMode;
boolean wandering; // after eating
boolean inPosition; // for attack
boolean inPursuit; // pursuing prey
Point startPursuit;
Point endPursuit;
```

```
int startTurn;
int endTurn;

public Predator(Environment e, int id, Predator lead, int mode) {
  super(e, id);
  leadPred = lead;
  opMode = mode;
  if (leadPred == null) {
    CurrLocation = new Point((int)(Math.random() * 50) - 25 + 400,
                             (int)(Math.random() * 50) - 25 + 400);
  }
  else
    CurrLocation = new Point(leadPred.getLocation());
  TargetLocation = new Point(CurrLocation);
  dirLineEndPt = new Point(CurrLocation);
  size = 12;
  maxSize = 20;
  visionRange = 140;
  switch ((int)(Math.random() * 3))
  {
    case 0: skinColor = Color.magenta;
            break;
    case 1: skinColor = Color.darkGray;
            break;
    case 2: skinColor = Color.red;
            break;
  }
  TargetPrey = null;
  wandering = true;
  inPosition = false;
  inPursuit = false;
  speed = 3; // lower is faster
  normalSpeed = 3;
  maxSpeed = 1;
} // end Predator(...)

public boolean positioned() {
  return inPosition;
}

public Prey getTargetPrey() {
  return TargetPrey;
}

public void performActivities(int turnNum) {
  if ((size != maxSize) && ((turnNum % 500) == 0)) {
    size++;
  }
  if ((turnNum == 0) && (leadPred == null)) {
    TargetLocation.x = CurrLocation.x + randDist();
    TargetLocation.y = CurrLocation.y + randDist();
    environ.adjustCoord(TargetLocation, 100);
  }
  if (turnNum == moveTurn){
    if ((TargetPrey == null) && wandering) {
      performWandering();
    }
    else if (TargetPrey == null) {
      seekPrey();
    }
    else {
      if (TargetPrey.isCaptured() || TargetPrey.inSafeZone()) {
        if ((leadPred == null) && inPursuit) {
          endPursuit = TargetPrey.getLocation();
          endTurn = environ.getTurnNumber();
          printPreyStats();
        }
        speed = normalSpeed;
        TargetPrey = null;
        wandering = true;
        inPosition = false;
        inPursuit = false;
        if (leadPred == null) {
          TargetLocation.x = CurrLocation.x + randDist();
          TargetLocation.y = CurrLocation.y + randDist();
          environ.adjustLeadCoord(TargetLocation, 100);
        } // end if (leadPred == null)
      }
      else if (CurrLocation.equals(TargetPrey.getLocation())) {
        TargetPrey.setCaptured();
      }
      else {
        pursuePrey();
      } // end else of if (TargetPrey.isCaptured())
    } // end else of if (TargetPrey == null)
    move();
    moveTurn = turnNum + speed;
  } // end if (turnNum == moveTurn)
} // end performActivities()

public void printPreyStats() {
  System.out.print("Py" + TargetPrey.getCreatureID());
  if (TargetPrey.isCaptured())
    System.out.print(" captured");
  else
    System.out.print(" escaped ");
  double dist = environ.computeDist(startPursuit, endPursuit);
  int turns = endTurn - startTurn;
  System.out.println("\tTurns: " + turns + "\tDist: " + dist);
} // end printPreyStats()

public void performWandering() {
  if (leadPred == null) {
    if (CurrLocation.equals(TargetLocation))
      seekPrey();
  }
  else {
    seekPrey(); // check to see if lead predator chose prey
    if (wandering) {
      moveRelativeToLead();
    } // end if (wandering)
  }
} // end performWandering()
```

```
public void moveRelativeToLead() {
  Point posLeadDir = leadPred.getDirPoint();
  Point posLeadPred = leadPred.getLocation();
  double tanOfAngle, angleInRad, angleInDeg;
  if ((posLeadDir.x - posLeadPred.x) == 0) {
    if (posLeadDir.y < posLeadPred.y)
      angleInDeg = -90.0;
    else
      angleInDeg = 90.0;
  }
  else {
    tanOfAngle = (double)(posLeadDir.y - posLeadPred.y) /
                 (double)(posLeadDir.x - posLeadPred.x);
    angleInRad = Math.atan(tanOfAngle);
    angleInDeg = Math.toDegrees(angleInRad);
  }
  if (posLeadDir.x < posLeadPred.x)
    angleInDeg = -180.0 + angleInDeg;
  double moveAngleInDeg = 0;
  if (creatureID == 1) {
    moveAngleInDeg = angleInDeg - 130.0;
  }
  else if (creatureID == 2) {
    moveAngleInDeg = angleInDeg + 130.0;
  }
  // compute desired location to follow the lead predator
  double moveAngleInRad = Math.toRadians(moveAngleInDeg);
  int xChange = (int) (30.0 * Math.cos(moveAngleInRad));
  int yChange = (int) (30.0 * Math.sin(moveAngleInRad));
  TargetLocation.x = posLeadPred.x + xChange;
  TargetLocation.y = posLeadPred.y + yChange;
  environ.adjustCoord(TargetLocation, 100);
  double dist = environ.computeDist(CurrLocation, posLeadPred);
  if (dist > 35.0)
    speed = maxSpeed + 1;
  else
    speed = normalSpeed;
} // end moveRelativeToLead()

public void seekPrey() {
  if (leadPred != null) {
    TargetPrey = leadPred.getTargetPrey();
    if (TargetPrey == null) {
      wandering = true;
    }
    else {
      wandering = false;
    }
  }
  else { // lead predator chooses prey
    LinkedList preyWithinRange =
        environ.givePreyInRange(CurrLocation, visionRange);
    ListIterator li = preyWithinRange.listIterator();
    double minDist = 999999; // large distance
    double dist = 0;
    Prey currPrey;
    while (li.hasNext()) {
      currPrey = (Prey) li.next();
      dist = environ.computeDist(CurrLocation, currPrey.getLocation());
      if ((dist < minDist) && !currPrey.inSafeZone()) {
        minDist = dist;
        TargetPrey = currPrey;
      }
    } // end while (li.hasNext())
    if (TargetPrey != null) {
      speed = maxSpeed;
      TargetLocation = TargetPrey.getLocation();
      wandering = false;
    }
    else {
      TargetLocation.x = CurrLocation.x + randDist();
      TargetLocation.y = CurrLocation.y + randDist();
      environ.adjustLeadCoord(TargetLocation, 100);
      wandering = true;
    }
  } // end else of if (leadPred != null)
} // end seekPrey()

public void pursuePrey() {
  switch (opMode) {
  case 1: simplePursuit();
          break;
  case 2: flankAndConverge();
          break;
  }
} // end pursuePrey()

public int findNumPredInPos() {
  LinkedList predList = environ.predList;
  ListIterator li = predList.listIterator();
  int numPredInPos = 0;
  while (li.hasNext()) {
    Predator currPred = (Predator) li.next();
    if (currPred.positioned()) {
      numPredInPos++;
    }
  } // end while
  return numPredInPos;
} // end findNumPredInPos()

public void simplePursuit() {
  double distToPrey = 0, distBtwnLeadPrey = 0;
  distToPrey = environ.computeDist(CurrLocation,
                                   TargetPrey.getLocation());
  if (leadPred != null) {
    distBtwnLeadPrey = environ.computeDist(leadPred.getLocation(),
                                        TargetPrey.getLocation());
  }
  if (inPursuit) {
    if ((leadPred != null) && (distToPrey < distBtwnLeadPrey))
      speed = maxSpeed + 1;
    else
      speed = maxSpeed;
  }
  else {
    inPursuit = true;
    TargetPrey.setBeingPursued();
    if (leadPred == null) {
      startPursuit = TargetPrey.getLocation();
      startTurn = environ.getTurnNumber();
    }
  }
  TargetLocation = TargetPrey.getLocation();
} // end simplePursuit

public void flankAndConverge() {
  double distToPrey = 0, distBtwnLeadPrey = 0;
  distToPrey = environ.computeDist(CurrLocation,
                      TargetPrey.getLocation());
  if (leadPred != null) {
    distBtwnLeadPrey = environ.computeDist(leadPred.getLocation(),
                      TargetPrey.getLocation());
  }
  if (inPursuit) {
    if ((leadPred != null) && (distToPrey < distBtwnLeadPrey))
      speed = maxSpeed + 1;
    else
      speed = maxSpeed;
    TargetLocation = TargetPrey.getLocation();
  }
  else {
    if (inPosition) {
      int numInPos = findNumPredInPos();
      if (numInPos == 3) {
        inPursuit = true;
        TargetPrey.setBeingPursued();
        if (leadPred == null) {
          startPursuit = TargetPrey.getLocation();
          startTurn = environ.getTurnNumber();
        }
      }
      if ((leadPred != null) && !inPursuit && (distToPrey > 90)) {
        inPosition = false;
      }
      else if (leadPred == null) {
        if (distToPrey > 80)
          TargetLocation = TargetPrey.getLocation();
        else
          TargetLocation = new Point(CurrLocation);
      }
    }
    else {
      determineFlankingPos();
    } // end else of if (inPosition)
  } // end else of if (inPursuit)
} // end flankAndConverge()

public void determineFlankingPos() {
  speed = normalSpeed - 1; // increase speed a little
  double distToPrey =
       environ.computeDist(CurrLocation, TargetPrey.getLocation());
  if (leadPred == null) {
    if (distToPrey < 80) {
      inPosition = true;
      TargetLocation = new Point(CurrLocation);
    }
    else {
      TargetLocation = TargetPrey.getLocation();
    }
  }
  else {
    if (leadPred.positioned()) {
      calcFlankLocation();
    }
    else {
      moveRelativeToLead();
    }
  } // end else of if (leadPred == null)
} // end determineFlankingPos()

public void calcFlankLocation() {
  Point posPrey = TargetPrey.getLocation();
  Point posLead = leadPred.getLocation();
  double tanOfAngle, angleInRad, angleInDeg;
  if ((posPrey.x - posLead.x) == 0) {
    if (posPrey.y < posLead.y)
      angleInDeg = -90.0;
    else
      angleInDeg = 90.0;
  }
  else {
    tanOfAngle =
        (double)(posPrey.y-posLead.y) / (double)(posPrey.x-posLead.x);
    angleInRad = Math.atan(tanOfAngle);
    angleInDeg = Math.toDegrees(angleInRad);
  }
  if (posPrey.x < posLead.x)
    angleInDeg = -180.0 + angleInDeg;
  double predAngleInDeg = 0;
  double predAngleInRad = 0;
  int xChange = 0, yChange = 0;
  double distToLead = environ.computeDist(CurrLocation, posLead);
  if (distToLead < 50) {
    if (creatureID == 1) {
      predAngleInDeg = angleInDeg - 90.0;
    }
    else if (creatureID == 2) {
      predAngleInDeg = angleInDeg + 90.0;
    }
    predAngleInRad = Math.toRadians(predAngleInDeg);
    xChange = (int) (100.0 * Math.cos(predAngleInRad));
    yChange = (int) (100.0 * Math.sin(predAngleInRad));
  }
  else {
    if (creatureID == 1) {
      predAngleInDeg = angleInDeg - 50.0;
    }
    else if (creatureID == 2) {
```

```
      predAngleInDeg = angleInDeg + 50.0;
    }
    predAngleInRad = Math.toRadians(predAngleInDeg);
    xChange = (int) (122.0 * Math.cos(predAngleInRad));
    yChange = (int) (122.0 * Math.sin(predAngleInRad));
  }
  // compute desired flanking location
  TargetLocation.x = leadPred.getLocation().x + xChange;
  TargetLocation.y = leadPred.getLocation().y + yChange;
  environ.adjustCoord(TargetLocation, 100);
  // check if approximately in position
  int diffx = CurrLocation.x - TargetLocation.x;
  int diffy = CurrLocation.y - TargetLocation.y;
  if ((diffx > -4 && diffx < 4) && (diffy > -4 && diffy < 4)) {
    inPosition = true;
    TargetLocation = new Point(CurrLocation);
  }
} // end calcFlankLocation()

public void draw(Graphics g) {
  g.setColor(skinColor);
  g.fillOval(CurrLocation.x - (size/2), CurrLocation.y - (size/2),
            size, size);
  // direction line
  g.setColor(Color.black);
  g.drawLine(CurrLocation.x, CurrLocation.y,
            dirLineEndPt.x, dirLineEndPt.y);
  g.drawString("   Pd" + creatureID, CurrLocation.x, CurrLocation.y);
  g.setColor(skinColor);
  g.drawOval(CurrLocation.x - visionRange, CurrLocation.y - visionRange,
            visionRange*2, visionRange*2);
} // end draw()

} // end class Predator
```

APPENDIX B
RAW DATA FROM COMPUTATIONAL EXPERIMENTS

The output from running the predator-prey program is listed below. The program was executed 20 times for each mode with 20,000 turns for each execution. The listing consists of the prey number, whether it was captured or escaped, the number of turns the prey was pursued, and the distance traveled from starting position to ending position once pursuit began.

Simple Pursuit Data

```
Mode: Simple Pursuit
Py0 captured      Turns: 159  Dist: 84.69356528095862
Py7 captured      Turns: 194  Dist: 82.71033792700887
Py6 captured      Turns: 221  Dist: 88.56635930193812
Py4 captured      Turns: 168  Dist: 100.42410069301094
Py8 captured      Turns: 227  Dist: 80.05623023850174
Py1 escaped       Turns: 112  Dist: 79.19595949289332
Py2 escaped       Turns: 124  Dist: 73.87827826905551
Py9 escaped       Turns: 72   Dist: 18.439088914585774
Py5 escaped       Turns: 139  Dist: 98.99494936611666

Mode: Simple Pursuit
Py2 captured      Turns: 182  Dist: 82.87339742040264
Py3 captured      Turns: 206  Dist: 59.53990258641679
Py6 captured      Turns: 227  Dist: 91.0
Py7 captured      Turns: 189  Dist: 102.95630140987001
Py0 captured      Turns: 158  Dist: 101.41498903022176
Py1 escaped       Turns: 44   Dist: 11.180339887498949
Py5 captured      Turns: 185  Dist: 95.0789145920377

Mode: Simple Pursuit
Py0 captured      Turns: 188  Dist: 88.05112151472007
Py6 escaped       Turns: 115  Dist: 63.00793600809346
Py9 captured      Turns: 152  Dist: 105.3612832116238
Py1 captured      Turns: 225  Dist: 80.75270893288967
Py8 captured      Turns: 160  Dist: 69.83552104767315
Py4 captured      Turns: 201  Dist: 91.35097153287424
Py7 captured      Turns: 165  Dist: 92.09777413162601
Py5 captured      Turns: 175  Dist: 103.81233067415451
Py2 escaped       Turns: 199  Dist: 80.06247560499239
Py3 captured      Turns: 210  Dist: 96.46242791885346
```

```
Mode: Simple Pursuit
Py5 captured      Turns: 221  Dist: 102.48902380255166
Py8 captured      Turns: 162  Dist: 79.1580697086532
Py1 captured      Turns: 201  Dist: 91.0494371207203
Py2 captured      Turns: 235  Dist: 107.48953437428223
Py0 captured      Turns: 171  Dist: 74.73285756613352
Py6 captured      Turns: 225  Dist: 91.98369420718001
Py3 captured      Turns: 180  Dist: 68.8839603971781
Py7 captured      Turns: 155  Dist: 77.02596964660685
Py9 captured      Turns: 180  Dist: 99.72963451251589
Py4 captured      Turns: 232  Dist: 108.25894882179486

Mode: Simple Pursuit
Py1 captured      Turns: 196  Dist: 83.24061508662703
Py6 captured      Turns: 247  Dist: 93.86160024205851
Py7 captured      Turns: 182  Dist: 104.80458005259122
Py4 captured      Turns: 190  Dist: 113.1591799192624
Py2 captured      Turns: 190  Dist: 108.90821823902914
Py5 escaped       Turns: 147  Dist: 61.91122676865643
Py3 captured      Turns: 139  Dist: 88.41379982785493
Py9 captured      Turns: 211  Dist: 94.92101980067429
Py0 captured      Turns: 216  Dist: 88.11923740024082
Py8 captured      Turns: 208  Dist: 74.46475676452586

Mode: Simple Pursuit
Py9 captured      Turns: 168  Dist: 99.29753269845128
Py3 captured      Turns: 159  Dist: 84.62860036654276
Py4 captured      Turns: 220  Dist: 95.88013350011565
Py0 captured      Turns: 238  Dist: 80.0812087820857
Py8 captured      Turns: 208  Dist: 109.59014554237986
Py6 captured      Turns: 194  Dist: 97.94386147176351
Py1 captured      Turns: 216  Dist: 58.077534382926416
Py5 captured      Turns: 204  Dist: 61.07372593840988
Py2 captured      Turns: 189  Dist: 81.00617260431454
Py7 escaped       Turns: 70   Dist: 43.139309220245984

Mode: Simple Pursuit
Py4 captured      Turns: 206  Dist: 98.47842403288143
Py6 captured      Turns: 182  Dist: 104.63746938836012
Py8 captured      Turns: 210  Dist: 104.21612159354233
Py3 captured      Turns: 204  Dist: 91.08786966440702
Py5 captured      Turns: 183  Dist: 104.80935072788114
Py7 captured      Turns: 225  Dist: 91.7877987534291
Py9 captured      Turns: 161  Dist: 86.6083136886985
Py0 captured      Turns: 224  Dist: 75.69015788066504
Py1 escaped       Turns: 59   Dist: 15.033296378372908
Py2 escaped       Turns: 157  Dist: 45.09988913511872

Mode: Simple Pursuit
Py4 captured      Turns: 191  Dist: 63.06346010171025
Py8 captured      Turns: 168  Dist: 80.0812087820857
Py2 escaped       Turns: 114  Dist: 63.89053137985315
Py3 escaped       Turns: 133  Dist: 94.75230867899737
Py5 captured      Turns: 215  Dist: 114.63420083029322
Py1 captured      Turns: 232  Dist: 104.23531071570709
Py6 captured      Turns: 158  Dist: 72.83543093852057
```

```
Py0 captured      Turns: 235  Dist: 90.4267659490264
Py7 captured      Turns: 152  Dist: 101.82337649086284
Py9 captured      Turns: 164  Dist: 107.71258050942797

Mode: Simple Pursuit
Py9 captured      Turns: 168  Dist: 103.00485425454472
Py1 captured      Turns: 198  Dist: 88.81441324469807
Py6 captured      Turns: 172  Dist: 100.70253224224304
Py8 captured      Turns: 230  Dist: 79.84985911070852
Py5 escaped       Turns: 71   Dist: 50.91168824543142
Py0 captured      Turns: 190  Dist: 102.08329931972223
Py7 captured      Turns: 200  Dist: 102.0
Py3 captured      Turns: 188  Dist: 87.20665112249179
Py4 captured      Turns: 269  Dist: 112.01785571952357
Py2 captured      Turns: 192  Dist: 84.71717653463199

Mode: Simple Pursuit
Py2 captured      Turns: 163  Dist: 102.7277956543408
Py5 captured      Turns: 180  Dist: 108.81176406988355
Py1 captured      Turns: 177  Dist: 104.23531071570709
Py8 escaped       Turns: 135  Dist: 96.16652224137046
Py7 captured      Turns: 180  Dist: 99.84988733093293
Py4 captured      Turns: 157  Dist: 94.02127418834527
Py3 captured      Turns: 190  Dist: 76.2954782408499
Py9 captured      Turns: 224  Dist: 90.79647570252934

Mode: Simple Pursuit
Py5 captured      Turns: 157  Dist: 98.85848471426213
Py6 captured      Turns: 214  Dist: 96.02083107326243
Py0 captured      Turns: 149  Dist: 85.70297544426332
Py9 captured      Turns: 240  Dist: 108.52188719332152
Py4 captured      Turns: 185  Dist: 102.95630140987001
Py7 escaped       Turns: 53   Dist: 38.18376618407357
Py1 captured      Turns: 264  Dist: 110.3177229641729
Py3 captured      Turns: 205  Dist: 82.87339742040264
Py8 captured      Turns: 203  Dist: 106.60675400742676
Py2 captured      Turns: 257  Dist: 102.07840124139877

Mode: Simple Pursuit
Py8 captured      Turns: 193  Dist: 105.152270541344
Py7 escaped       Turns: 65   Dist: 46.66904755831214
Py1 captured      Turns: 228  Dist: 90.21086409075129
Py2 captured      Turns: 210  Dist: 86.1278120005379
Py0 captured      Turns: 224  Dist: 90.0222194794152
Py6 captured      Turns: 213  Dist: 85.44003745317531
Py4 escaped       Turns: 107  Dist: 32.31098884280702
Py3 captured      Turns: 228  Dist: 77.47257579298626
Py9 captured      Turns: 175  Dist: 103.81233067415451
Py5 escaped       Turns: 110  Dist: 62.625873247404705

Mode: Simple Pursuit
Py9 captured      Turns: 186  Dist: 95.2732911156112
Py2 captured      Turns: 145  Dist: 85.44588931013593
Py7 escaped       Turns: 98   Dist: 43.266615305567875
Py6 captured      Turns: 271  Dist: 107.20074626605917
Py5 captured      Turns: 228  Dist: 104.17293314484334
Py8 escaped       Turns: 6    Dist: 4.242640687119285
```

```
Py4 captured      Turns: 175  Dist: 64.25729530566937
Py3 captured      Turns: 232  Dist: 99.04039579888602
Py0 captured      Turns: 192  Dist: 104.21612159354233
Py1 captured      Turns: 252  Dist: 109.03669107231748

Mode: Simple Pursuit
Py5 captured      Turns: 181  Dist: 102.07840124139877
Py3 captured      Turns: 228  Dist: 97.41663102366043
Py6 escaped       Turns: 95   Dist: 19.79898987322333
Py9 escaped       Turns: 66   Dist: 30.23243291566195
Py2 captured      Turns: 149  Dist: 88.72992730753249
Py8 escaped       Turns: 109  Dist: 77.78174593052023
Py4 captured      Turns: 182  Dist: 84.09518416651456
Py1 captured      Turns: 219  Dist: 115.8792474949678
Py0 captured      Turns: 223  Dist: 80.77747210701756
Py7 captured      Turns: 178  Dist: 80.23091673413684

Mode: Simple Pursuit
Py7 captured      Turns: 172  Dist: 94.37160589923221
Py2 captured      Turns: 164  Dist: 102.82509421342633
Py1 captured      Turns: 221  Dist: 93.5200513259055
Py0 captured      Turns: 147  Dist: 84.86459803710851
Py4 captured      Turns: 151  Dist: 67.36467917239716
Py6 captured      Turns: 181  Dist: 62.433965115151864
Py5 captured      Turns: 231  Dist: 77.6659513557904
Py8 escaped       Turns: 48   Dist: 12.041594578792296

Mode: Simple Pursuit
Py3 captured      Turns: 153  Dist: 83.54639429682169
Py5 escaped       Turns: 99   Dist: 70.71067811865476
Py6 captured      Turns: 164  Dist: 102.59142264341595
Py0 captured      Turns: 214  Dist: 93.05912099305473
Py1 captured      Turns: 226  Dist: 115.00434774390054
Py8 captured      Turns: 224  Dist: 88.81441324469807

Mode: Simple Pursuit
Py8 captured      Turns: 197  Dist: 82.22530024268686
Py3 captured      Turns: 201  Dist: 106.6255128944288
Py9 captured      Turns: 150  Dist: 71.17583859709698
Py7 escaped       Turns: 37   Dist: 19.849433241279208
Py4 escaped       Turns: 12   Dist: 1.4142135623730951
Py1 captured      Turns: 192  Dist: 98.49365461794989
Py5 escaped       Turns: 139  Dist: 62.609903369994115
Py0 captured      Turns: 261  Dist: 110.76551810017412
Py6 captured      Turns: 225  Dist: 83.21658488546619
Py2 captured      Turns: 175  Dist: 111.66467659918243

Mode: Simple Pursuit
Py0 captured      Turns: 153  Dist: 71.84010022264724
Py4 escaped       Turns: 221  Dist: 75.92759709091287
Py3 captured      Turns: 147  Dist: 63.953107821277925
Py9 captured      Turns: 172  Dist: 93.91485505499116
Py2 captured      Turns: 163  Dist: 91.21403400793103
Py5 captured      Turns: 207  Dist: 100.08995953640904
Py8 escaped       Turns: 178  Dist: 108.2450922675019
Py7 escaped       Turns: 153  Dist: 77.15568676384133
Py6 escaped       Turns: 38   Dist: 5.656854249492381
```

```
Py1 escaped       Turns: 26   Dist: 7.0710678118654755

Mode: Simple Pursuit
Py7 captured      Turns: 154  Dist: 79.81227975693966
Py5 escaped       Turns: 73   Dist: 52.32590180780452
Py8 captured      Turns: 223  Dist: 119.4026800369238
Py2 captured      Turns: 224  Dist: 92.02173656261873
Py0 escaped       Turns: 17   Dist: 8.602325267042627
Py6 captured      Turns: 187  Dist: 81.93289937503738
Py9 captured      Turns: 183  Dist: 97.49871794028884
Py4 escaped       Turns: 90   Dist: 26.1725046566048
Py1 captured      Turns: 193  Dist: 70.83078426785913
Py3 escaped       Turns: 95   Dist: 31.400636936215164

Mode: Simple Pursuit
Py9 captured      Turns: 155  Dist: 86.95401083331349
Py8 escaped       Turns: 98   Dist: 50.24937810560445
Py0 escaped       Turns: 103  Dist: 65.19202405202648
Py5 captured      Turns: 218  Dist: 80.62257748298549
Py1 captured      Turns: 162  Dist: 104.35037134576953
Py6 captured      Turns: 164  Dist: 91.21403400793103
Py7 escaped       Turns: 130  Dist: 89.10667763978185
Py4 escaped       Turns: 145  Dist: 80.0
Py3 captured      Turns: 152  Dist: 96.16652224137046
Py2 captured      Turns: 191  Dist: 80.72174428244226

Mode: Simple Pursuit
Py1 captured      Turns: 162  Dist: 100.65783625729296
Py7 escaped       Turns: 107  Dist: 21.540659228538015
Py8 escaped       Turns: 80   Dist: 21.02379604162864
Py0 captured      Turns: 205  Dist: 105.58882516630251
Py6 captured      Turns: 173  Dist: 89.69392398596462
Py5 captured      Turns: 172  Dist: 102.34256201600583
Py3 escaped       Turns: 87   Dist: 53.74011537017761
Py9 captured      Turns: 162  Dist: 83.6301381082203
Py4 captured      Turns: 230  Dist: 95.08417323613851
Py2 captured      Turns: 176  Dist: 72.62231062146122
```

Flank and Converge Data

```
Mode: Flank & Converge
Py8 captured      Turns: 126  Dist: 62.00806399170998
Py5 captured      Turns: 150  Dist: 74.06078584514209
Py7 captured      Turns: 140  Dist: 64.28063471995279
Py4 escaped       Turns: 76   Dist: 45.254833995939045
Py6 captured      Turns: 114  Dist: 60.108235708594876
Py2 captured      Turns: 152  Dist: 70.57619995437555
Py1 captured      Turns: 102  Dist: 45.79301256742124
Py0 escaped       Turns: 95   Dist: 56.302753041036986
Py3 captured      Turns: 96   Dist: 34.88552708502482

Mode: Flank & Converge
Py6 captured      Turns: 108  Dist: 64.66065264130884
Py3 captured      Turns: 109  Dist: 32.14031735997639
Py0 captured      Turns: 138  Dist: 64.8459713474939
```

```
Py7 captured      Turns: 134  Dist: 69.1158447825099
Py2 captured      Turns: 147  Dist: 67.47592163134935
Py9 captured      Turns: 92   Dist: 55.17245689653489
Py5 captured      Turns: 131  Dist: 78.56844150166147
Py1 captured      Turns: 123  Dist: 68.41052550594829
Py4 captured      Turns: 143  Dist: 71.11258679024411
Py8 captured      Turns: 130  Dist: 36.6742416417845

Mode: Flank & Converge
Py2 captured      Turns: 87   Dist: 3.1622776601683795
Py1 escaped       Turns: 13   Dist: 2.23606797749979
Py3 captured      Turns: 130  Dist: 80.15609770940699
Py5 captured      Turns: 134  Dist: 83.43860018001261
Py0 captured      Turns: 151  Dist: 70.06425622241343
Py4 escaped       Turns: 61   Dist: 31.016124838541646
Py6 captured      Turns: 100  Dist: 43.01162633521314
Py9 captured      Turns: 113  Dist: 67.89698078707183
Py7 escaped       Turns: 73   Dist: 45.254833995939045
Py8 captured      Turns: 138  Dist: 82.75868534480233

Mode: Flank & Converge
Py1 captured      Turns: 90   Dist: 29.068883707497267
Py6 captured      Turns: 105  Dist: 52.80151512977634
Py9 captured      Turns: 140  Dist: 66.007575322837
Py4 captured      Turns: 142  Dist: 81.83520025025906
Py8 captured      Turns: 150  Dist: 69.02897942168927
Py0 captured      Turns: 92   Dist: 37.589892258425
Py2 captured      Turns: 110  Dist: 52.773099207835045
Py7 captured      Turns: 142  Dist: 76.8505042273634
Py3 captured      Turns: 131  Dist: 84.50443775329198
Py5 captured      Turns: 109  Dist: 20.12461179749811

Mode: Flank & Converge
Py8 captured      Turns: 137  Dist: 66.12866246946176
Py1 captured      Turns: 129  Dist: 64.4980619863884
Py4 captured      Turns: 158  Dist: 79.83107164506812
Py3 captured      Turns: 158  Dist: 93.72299611087985
Py6 captured      Turns: 112  Dist: 39.0
Py5 captured      Turns: 95   Dist: 42.20189569201838
Py0 escaped       Turns: 153  Dist: 75.0066663703967
Py7 escaped       Turns: 73   Dist: 33.97057550292606
Py2 captured      Turns: 140  Dist: 62.20128616033595
Py9 escaped       Turns: 120  Dist: 77.10382610480494

Mode: Flank & Converge
Py6 captured      Turns: 125  Dist: 64.53681120105021
Py1 escaped       Turns: 28   Dist: 18.384776310850235
Py2 captured      Turns: 133  Dist: 81.32035415564789
Py7 captured      Turns: 106  Dist: 32.55764119219941
Py8 captured      Turns: 106  Dist: 25.495097567963924
Py9 captured      Turns: 147  Dist: 84.02380615040002
Py0 captured      Turns: 124  Dist: 61.07372593840988
Py5 captured      Turns: 118  Dist: 70.03570517957252
Py4 escaped       Turns: 72   Dist: 42.44997055358225
Py3 captured      Turns: 149  Dist: 67.11929677819934

Mode: Flank & Converge
```

```
Py8 captured      Turns: 107  Dist: 11.40175425099138
Py0 captured      Turns: 95   Dist: 34.539832078341085
Py3 escaped       Turns: 30   Dist: 10.816653826391969
Py6 captured      Turns: 113  Dist: 45.12205669071391
Py4 captured      Turns: 157  Dist: 87.80091115700337
Py9 captured      Turns: 139  Dist: 69.06518659932803
Py2 captured      Turns: 151  Dist: 88.09086218218096
Py5 captured      Turns: 101  Dist: 37.013511046643494
Py7 captured      Turns: 113  Dist: 33.421549934136806

Mode: Flank & Converge
Py8 escaped       Turns: 100  Dist: 40.607881008493905
Py7 captured      Turns: 112  Dist: 55.00909015790027
Py3 captured      Turns: 133  Dist: 84.86459803710851
Py2 captured      Turns: 101  Dist: 36.138621999185304
Py0 captured      Turns: 91   Dist: 18.027756377319946
Py9 captured      Turns: 122  Dist: 77.07788269017254
Py5 captured      Turns: 141  Dist: 64.76109943476871
Py4 captured      Turns: 92   Dist: 31.953090617340916
Py6 escaped       Turns: 115  Dist: 69.35416353759881
Py1 captured      Turns: 129  Dist: 82.03657720797473

Mode: Flank & Converge
Py7 captured      Turns: 139  Dist: 65.7875368135941
Py8 escaped       Turns: 82   Dist: 41.048751503547585
Py6 captured      Turns: 94   Dist: 42.05948168962618
Py4 captured      Turns: 134  Dist: 82.07313811473277
Py9 captured      Turns: 116  Dist: 57.42821606144492
Py1 captured      Turns: 122  Dist: 70.49113419430843
Py3 captured      Turns: 117  Dist: 57.38466694161429
Py0 captured      Turns: 135  Dist: 82.13403679352427
Py2 escaped       Turns: 47   Dist: 27.018512172212592
Py5 captured      Turns: 95   Dist: 27.0

Mode: Flank & Converge
Py6 captured      Turns: 131  Dist: 79.2464510246358
Py4 escaped       Turns: 138  Dist: 74.65252842335616
Py7 captured      Turns: 115  Dist: 43.56604182158393
Py0 captured      Turns: 89   Dist: 46.69047011971501
Py3 captured      Turns: 107  Dist: 60.8276253029822
Py9 captured      Turns: 130  Dist: 82.73451516749222
Py8 captured      Turns: 137  Dist: 82.73451516749222
Py5 escaped       Turns: 66   Dist: 46.66904755831214

Mode: Flank & Converge
Py4 captured      Turns: 142  Dist: 61.204574992397426
Py6 captured      Turns: 89   Dist: 18.788294228055936
Py0 captured      Turns: 150  Dist: 68.0
Py1 escaped       Turns: 109  Dist: 70.09279563550022
Py7 escaped       Turns: 124  Dist: 82.03657720797473
Py2 captured      Turns: 138  Dist: 56.08029957123981
Py5 captured      Turns: 95   Dist: 27.202941017470888
Py3 captured      Turns: 132  Dist: 77.10382610480494
Py9 escaped       Turns: 64   Dist: 30.479501308256342
Py8 captured      Turns: 142  Dist: 71.58910531638176

Mode: Flank & Converge
```

```
Py4 captured      Turns: 138  Dist: 79.2464510246358
Py0 captured      Turns: 100  Dist: 57.28001396647874
Py8 captured      Turns: 160  Dist: 80.4114419718985
Py9 captured      Turns: 118  Dist: 73.0
Py2 captured      Turns: 137  Dist: 78.81624198095213
Py3 captured      Turns: 90   Dist: 22.47220505424423
Py5 captured      Turns: 107  Dist: 28.635642126552707
Py7 captured      Turns: 145  Dist: 88.40814442120137
Py1 captured      Turns: 106  Dist: 60.8276253029822
Py6 captured      Turns: 111  Dist: 36.359317925395686

Mode: Flank & Converge
Py7 captured      Turns: 148  Dist: 67.7421582177598
Py0 captured      Turns: 102  Dist: 58.412327466040935
Py8 captured      Turns: 134  Dist: 53.009433122794285
Py9 captured      Turns: 150  Dist: 71.06335201775947
Py1 captured      Turns: 116  Dist: 60.440052945046304
Py2 captured      Turns: 122  Dist: 37.013511046643494
Py5 captured      Turns: 101  Dist: 20.12461179749811
Py3 captured      Turns: 89   Dist: 19.0
Py6 captured      Turns: 135  Dist: 66.61080993352356

Mode: Flank & Converge
Py9 captured      Turns: 135  Dist: 58.42088667591412
Py0 captured      Turns: 140  Dist: 89.87213138676528
Py7 captured      Turns: 104  Dist: 29.068883707497267
Py4 captured      Turns: 160  Dist: 86.83893136145792
Py5 captured      Turns: 101  Dist: 22.360679774997898
Py3 captured      Turns: 96   Dist: 27.294688127912362
Py8 escaped       Turns: 82   Dist: 36.49657518178932
Py2 escaped       Turns: 115  Dist: 59.23681287847955
Py6 captured      Turns: 127  Dist: 71.56116265125938
Py1 captured      Turns: 91   Dist: 20.615528128088304

Mode: Flank & Converge
Py2 captured      Turns: 116  Dist: 60.0
Py9 captured      Turns: 104  Dist: 6.324555320336759
Py3 captured      Turns: 155  Dist: 91.43850392476902
Py7 captured      Turns: 102  Dist: 61.61980201201558
Py5 escaped       Turns: 48   Dist: 26.248809496813376
Py0 captured      Turns: 127  Dist: 71.11961754677819
Py6 captured      Turns: 107  Dist: 46.95742752749558
Py8 captured      Turns: 102  Dist: 28.460498941515414
Py4 captured      Turns: 104  Dist: 12.529964086141668
Py1 captured      Turns: 84   Dist: 19.4164878389476

Mode: Flank & Converge
Py4 captured      Turns: 89   Dist: 23.53720459187964
Py1 captured      Turns: 144  Dist: 64.93843238021688
Py5 captured      Turns: 111  Dist: 54.589376255824725
Py6 captured      Turns: 151  Dist: 69.72087205421343
Py3 escaped       Turns: 65   Dist: 34.828149534535996
Py2 captured      Turns: 137  Dist: 83.23460818673925
Py9 captured      Turns: 134  Dist: 76.02631123499285
Py7 captured      Turns: 157  Dist: 79.62411694957753
Py0 escaped       Turns: 41   Dist: 15.811388300841896
Py8 captured      Turns: 148  Dist: 75.58438992278762
```

```
Mode: Flank & Converge
Py1 captured      Turns: 140  Dist: 61.61168720299745
Py9 captured      Turns: 95   Dist: 11.704699910719626
Py3 captured      Turns: 115  Dist: 49.0
Py6 captured      Turns: 152  Dist: 70.71067811865476
Py8 captured      Turns: 86   Dist: 26.570660511172846
Py2 escaped       Turns: 42   Dist: 25.45584412271571
Py0 captured      Turns: 122  Dist: 72.83543093852057
Py5 escaped       Turns: 73   Dist: 40.45985664828782

Mode: Flank & Converge
Py0 captured      Turns: 127  Dist: 76.40026177965623
Py8 escaped       Turns: 36   Dist: 15.556349186104045
Py3 escaped       Turns: 54   Dist: 18.681541692269406
Py7 captured      Turns: 106  Dist: 42.579337712087536
Py1 captured      Turns: 125  Dist: 62.12889826803627
Py9 captured      Turns: 124  Dist: 77.78174593052023
Py4 captured      Turns: 150  Dist: 69.58448102845921
Py6 escaped       Turns: 31   Dist: 15.620499351813308
Py2 captured      Turns: 117  Dist: 49.72926703662542

Mode: Flank & Converge
Py5 captured      Turns: 110  Dist: 68.62215385719105
Py8 captured      Turns: 78   Dist: 42.5205832509386
Py2 captured      Turns: 108  Dist: 55.65968020030299
Py3 captured      Turns: 134  Dist: 62.03224967708329
Py9 captured      Turns: 132  Dist: 76.69419795525604
Py7 captured      Turns: 146  Dist: 67.1863081289633
Py6 captured      Turns: 123  Dist: 61.1310068623117
Py4 captured      Turns: 130  Dist: 84.21995013059554
Py1 captured      Turns: 110  Dist: 41.30375285612676
Py0 captured      Turns: 86   Dist: 31.575306807693888

Mode: Flank & Converge
Py1 captured      Turns: 151  Dist: 69.87131027825369
Py7 captured      Turns: 114  Dist: 45.880278987817846
Py5 captured      Turns: 129  Dist: 69.85699678629192
Py0 captured      Turns: 109  Dist: 54.644304369257
Py2 escaped       Turns: 121  Dist: 75.66372975210778
Py6 captured      Turns: 161  Dist: 80.89499366462674
Py9 captured      Turns: 108  Dist: 51.62363799656123
Py8 captured      Turns: 106  Dist: 65.94694837519019
Py3 escaped       Turns: 48   Dist: 29.698484809834994
Py4 captured      Turns: 138  Dist: 87.0
```

LIST OF REFERENCES

Bou99 Boutilier, Craig. "Multiagent Systems: Challenges and Opportunities for Decision-Theoretic Planning." AI Magazine. Winter 1999: 35-43.

Dug97 Dugatkin, Lee Alan. Cooperation Among Animals: An Evolutionary Perspective. New York: Oxford University Press, 1997.

Dur87 Durfee, Edmund H., Victor R. Lesser, and Daniel D. Corkill. "Coherent Cooperation among Communicating Problem Solvers." IEEE Transactions on Computers. C-36.11 (November 1987): 1275-1291.

Fun98 Funston, P. J., M. G. L. Mills, H. C. Biggs, and P. R. K. Richardson. "Hunting by Male Lions: Ecological Influences and Socioecological Implications." Animal Behaviour. 56: 1333-1345.

Hay98 Hayes-Roth, Barbara, Lee Brownston, and Robert van Gent. "Multiagent Collaboration in Directed Improvisation." Readings in Agents. Ed. Michael N. Huhns and Munindar P. Singh. San Francisco, Calif.: Morgan Kaufmann Publishers, Inc., 1998. 141-147.

Hei95 Heinsohn, Robert, and Craig Packer. "Complex Cooperative Strategies in Group-Territorial African Lions." Science. 269 (Sept. 1, 1995): 1260-1262.

Huh99 Huhns, Michael N., and Larry M. Stephens. "Multiagent Systems and Societies of Agents." Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Ed. Gerhard Weiss. Cambridge, Mass.: The MIT Press, 1999. 79-120.

Ken01 Kennedy, James, and Russell C. Eberhart. Swarm Intelligence. San Diego, Calif.: Academic Press, 2001.

Lan97 Lander, Susan E., and Victor R. Lesser. "Sharing Metainformation to Guide Cooperative Search Among Heterogeneous Reusable Agents." IEEE Transactions on Knowledge and Data Engineering. 9.2 (March-April 1997): 193-208.

Pac97 Packer, Craig, and Anne E. Pusey. "Divided We Fall: Cooperation Among Lions." Scientific American. May 1997: 52-59.

Par99 Parunak, H. Van Dyke. "Industrial and Practical Application of DAI." Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Ed. Gerhard Weiss. Cambridge, Mass.: The MIT Press, 1999. 377-421.

Sch91 Scheel, D., and C. Packer. "Group Hunting Behaviour of Lions: A Search for Cooperation." Animal Behaviour. 41 (April 1991): 697-709.

Sin94 Singh, Munindar P. Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications. Berlin: Springer-Verlag, 1994.

Smi95 Smithers, Tim. "Are Autonomous Agents Information Processing Systems?" The Artificial Life Route to Artificial Intelligence: Building Embodied, Situated Agents. Ed. Luc Steels and Rodney Brooks. Hillsdale, NJ: Lawrence Erlbaum Associates, 1995. 123-162.

Tes01 Tessier, Catherine, Heinz-Jurgen Muller, Humbert Fiorino, and Laurent Chaudron. "Agents' Conflicts: New Issues." Conflicting Agents: Conflict Management in Multi-Agent Systems. Ed. Catherine Tessier, Laurent Chaudron, and Heinz-Jurgen Muller. Norwell, Mass.: Kluwer Academic Press, 2001. 1-30.

Woo99 Wooldridge, Michael. "Intelligent Agents." Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Ed. Gerhard Weiss. Cambridge, Mass.: The MIT Press, 1999. 27-77.

BIOGRAPHICAL SKETCH

Before returning to college for graduate studies in fall 2000, Richard Bartleson spent more than five years as a software developer and senior software developer for Andersen Consulting (now known as Accenture). He worked in a *solution center* that focused on a customer service system used in the utilities industry. In addition to programming and design activities, Richard participated in continuous improvement efforts that helped the center achieve Level 3 based on the Software Engineering Institute's Capability Maturity Model. Also, he was a "learning center" coach, responsible for training newly hired developers.

In the early 1990s, Richard spent time as a regional chapters coordinator for the National Space Society, a non-profit organization promoting space development. In the mid- to late-1980s, he spent three and a half years as a Public Affairs Specialist in the U.S. Air Force, working as a staff writer for the base newspaper and media relations department.

Richard's undergraduate education consists of B.S. degrees in computer science (1995) and management (1990), both from the University of West Florida in Pensacola, Florida, as well as an A.A.S. degree in electronics technology (1984) from a community college in Virginia.