A SYSTEM FOR SYNTHESIZING SWARM PROGRAMS

A Thesis
in TCC402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Science

by

Errol Charles McEachron

March 26, 2002

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

_____
(Full signature)

Approved _____ (Technical Advisor)
Dave Evans (Signature)

Approved _____ (TCC Advisor)
Kathryn Neeley (Signature)

PREFACE

The original motivation for undertaking this project stemmed from my perpetual interests in the latest and greatest technologies. I first learned of swarm programming from a talk given by my technical advisor, Dave Evans. Professor Evans discussed some of the many applications of swarm technology, but it was the application to simulated soccer that first captivated my attention. After talking with Professor Evans, I learned that swarm programming at the University of Virginia was in its beginning stages and that there were many research opportunities available. During this time, I thought anything with the words swarm programming sounded interesting, but I was especially intrigued by the idea of automatically generating swarm programs based on some high-level description. This idea formed the basis for my research and concluded in the project presented by this technical report.

The development of this project was initially based on the work performed by the Swarm Development Group (SDG) in Santa Fe. SDG had already developed an infrastructure for creating and interpreting swarm programs. The swarm structure provided by SDG was initially implemented with the Objective C programming language, and later an interface was created for Java users, which somewhat complicated the semantics of the Java programming language. Other researchers expressed their displeasure with the Sante Fe system and one gentleman in particular, Mike Hogye, took it upon himself to begin developing an entirely new swarm framework. Mike Hogye implemented the swarm framework in C++ and designed it to use the Raptor Simulator, which is a general network simulator capable of simulating the dynamic network environments associated with swarm programming. Since the Raptor Simulator project is

located here, at the University of Virginia, it would be relatively easy to make modifications that would accommodate swarm development. Hence, the Swarm project was moved from the Santa Fe simulator to the Raptor simulator in late January. The creation of the swarm framework and the relocation of the project provided a more flexible approach for developing swarm programs.

I commend Mike Hogye for his extraordinary efforts in developing the swarm framework. I would also like to thank my technical advisor, Dave Evans for, first, providing me the opportunity to research swarm technology, and, secondly, for his time and suggestions that all contributed to the success of this project. Lastly, I express my sincerest appreciations to my TCC advisor, Kathryn Neeley, for her support and expert guidance in the development of this report. I hope that this project serves as a testament to the efforts of all those who helped along the way, and basis for all those to come in the future.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY OF TERMS

- ***abstraction*** **-** the process of focusing upon the essential characteristics of an object

- ***acceptable swarm program*** **–** a swarm program is acceptable if it exhibits the intended behavior during operation, for a reasonably large number of operations

- ***agent*** – a computational component with limited capabilities for maintaining and modifying internal data representations (memory or state) while interacting with the environment

- ***base class*** **–** the most generalized class in a class structure from which other classes are inherited

- ***behavior*** – a description of the global dynamics that emerge from the collected interactions of individual devices

- ***class*** **–** a set of objects that share a common structure and a common behavior (.h and .cpp files)

- ***derived class*** – a class that inherits from one or more generalized base classes

- ***device*** – synonymous with agent (see *agent*)

- ***environment*** – the dynamic physical context in which the swarm exists and operates

- ***generate*** **–** the process used by the swarm generator to create a swarm program

- ***information hiding*** **–** the process of keeping the implementation details of an object hidden

- ***inheritance*** – a relationship among classes, wherein one class shares the structure or behavior defined in one (single inheritance) or more (multiple inheritance) other classes. Inheritance defines an "is-a" hierarchy among classes in which a subclass, or derived class, inherits from one or more generalized base classes.

- ***instance*** **–** the creation of an object in the program

- ***member functions*** – an operation upon an object, defined as part of the declaration of a class

- ***object*** – defined by a class, it has state, behavior, and identity

- ***pure virtual function –*** a member function that must be defined by a derived class

- ***software system*** – synonomous with the swarm generator

- ***swarm*** – a collection of devices

- ***swarm framework*** – the swarm code library, or structure, that interfaces with the Raptor Simulator and provides the infrastructure for which this project was developed.

- ***swarm generator –*** a software class that accepts a high level behavioral descriptions and synthesizes an acceptable swarm application

- ***swarm program*** – the computer code that runs on an individual swarm device

- ***swarm programming*** – programming a collection of autonomous swarm devices

- ***synthesizing*** – the overall process of creating a swarm program from a set of high-level behaviors

ABSTRACT

Recently, advancements in computing technology have revolutionized the traditional concept of computer programming. Future programs will operate on collections of mobile processors that communicate over wireless networks and function in dynamic environments. These collections can be viewed as computational swarms, similar to those swarms found in nature, such as ants or bees. As with any swarm, its behavior emerges from the collective behaviors of its individual members. Thus, a swarm's behavior must be resilient to the misbehavior of a few individual members. This concept marks the fundamental difference between swarm programming and traditional programming.

Swarm programming requires a flexible system that can handle the dynamic nature of swarm environments and the random failure of swarm devices. This requirement makes swarm programming rather time consuming and quite tedious. This project addressed this issue by developing a software system to generate swarm programs from a set of high-level descriptions.

The software system was tested with an example search-and-rescue application, which includes the disperse, converge, and rescue swarm behaviors. Although the software system successfully assimilated the three behaviors into an acceptable swarm program, the test results indicate that the system may not handle all variations of the same swarm application equally well. Studying this issue and making any appropriate modifications to the software system could be an interesting application of future research.

CHAPTER 1: INTRODUCTION

Swarm programming is a process for creating computer programs to control collections of autonomous computing devices with limited individual resources. Presently, the process for programming these collections, or swarms, is time consuming and requires significant attention to detail. This project produced a software system to generate swarm programs from a set of high-level descriptions and, thereby, increased the efficiency associated with the swarm programming process.

1. PROGRAMMING THE SWARM

In the last decade, there have been major advancements in computing technology that will largely influence the design, implementation, and interpretation of computer programs in the near future. Programming will evolve to operate on clusters of autonomously distributed systems that communicate over ad hoc networks [Evans, 2000]. The inherent properties of such collections are similar to those of the biological swarms found in nature. Consider a colony of ants recovering food left over from a picnic, or a swarm of bees searching for an acceptable location for a new hive. In both situations, each insect is interacting with the environment and relaying information throughout the rest of the swarm. The result is the swarm functioning as a single entity in order to achieve a desired behavior. In this way, one can view a collection of interdependent, mobile, communicating devices as a swarm, and the notion of programming them as swarm programming.

The single most important concept governing swarm programming is that the behavior of the swarm emerges from the collective behavior of the individual devices [Evans, 2000]. This means that the behavior of the swarm must be resilient to the

misbehavior of a few individual devices. This fundamental concept will not only provide a basis for developing present swarm technologies, but will persist to govern the very complex applications of these technologies into the future.

For example, consider the distant application of swarm programming to the construction of a new bridge for a highway transportation system. The construction of a bridge by current methods is a monumental task that involves many engineering disciplines and many different people. However, in the future, swarm technologies could be used to build this same bridge in what amounts to the press of a single button. A simplified view of this goal is displayed in Figure 1 [Evans, 2000].

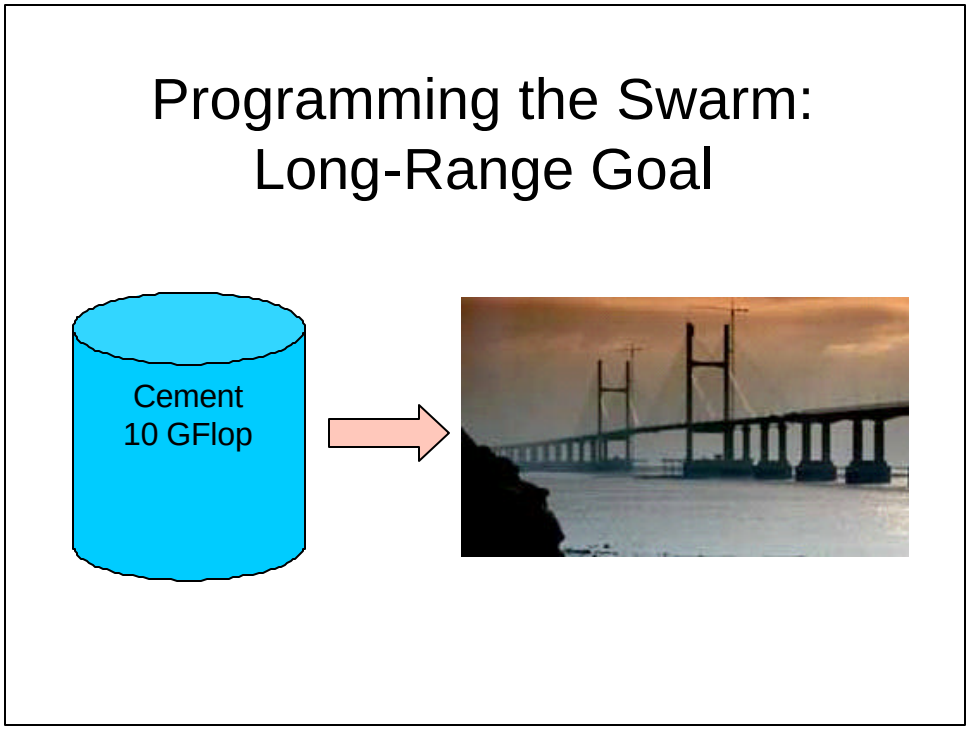

Figure 1. A Long-Range Goal of Swarm Programming: This represents one of the long-range goals of swarm programming. The idea is to achieve great complexity from collections of extreme simplicity. "Swarm Programming: How to Program a Micronet" http://www.cs.virginia.edu/~evans /talks/index.html

Instead of purchasing materials like concrete and steel beams, construction companies would purchase billions of swarm devices, which would serve as both the

materials and resources required to build the new bridge. Swarm devices could be constructed such that they fit together in the same manner that a bolt would fit a screw, thus, giving swarms a method for interconnection. Additionally, swarm devices could be mixed with materials like paint and concrete, which automates the use of these materials in construction. Each device would contain a swarm program that is responsible for governing the overall behavior of the swarm, just as a superintendent or site foreman would be responsible for coordinating the efforts of the company workers. In this sense, a swarm program is analogous to an architectural plan for a specific bridge, as well as the collective administration that dictates its construction. Hence, the procedure for constructing a bridge using swarm technology would require the devices to interact with one another and their environment based on a set of predefined rules provided by the swarm program. It is within the development of those rules that the difficulties of programming the swarm exist.

2. THE DIFFICULTIES ASSOCIATED WITH SWARM PROGRAMMING

The computing infrastructure for programming the swarm is based on many interdependent devices, each with the limited ability to communicate, process and store information, as well as change position [Evans, 2000]. The dynamic and unpredictable nature of such an infrastructure makes traditional programming methodologies inadequate for the use with swarm technology. While time and memory are the most limited resources in traditional programming, swarm programs require the adaptive management of a limited source of power allocated to device processing, communication, and mobility [Evans, 2000]. The efficient use of resources is a fundamental aspect of swarm programming. It will often require the approximation of a specific swarm

function in order to conserve resources and acceptably accomplish the intended overall objective. These inherent requirements of swarm programming have hindered the ability to efficiently create swarm programs and require the development of new programming methodologies.

Presently, the technology for designing and machining the computer hardware necessary to build swarms greatly exceeds our ability to program them in a useful manner. This follows from a lack of well-formed methodologies for designing, implementing, and reasoning about swarm programs. The design aspect of a swarm program is complicated by the numerous possibilities surrounding the formal definition of a swarm behavior. In other words, how does one formally describe a swarm's behavior in such a way that facilitates the development of swarm programs? Although a complete answer to this question is many years away, assume for a moment that adequate methodologies for formally defining a swarm behavior already exist. Then the next question and the focus of this project is:

Given a high-level description of a swarm's behavior can we generate an acceptable swarm program?

The answer to this question follows from the ability to determine what is acceptable for a given high-level swarm behavior and how to translate it into an acceptable swarm program. For the purposes of this project, an acceptable swarm program is one in which the intended behavior of the program emerges from the collective behaviors of the individual swarm devices an acceptable amount of the time. In other words, an acceptable swarm program is only required to exhibit the desired swarm behavior some acceptable percentage of the time. The development of a software program, or swarm

generator, addressed these questions by successfully generating an acceptable swarm program from a high-level behavioral description. This project began where the developed framework for reasoning about swarm programs left off, and concluded by improving the efficiency in relation to the development of such programs.

However, developing methodologies for reasoning about swarm programs is only part of the programming problem. Currently, many of the difficulties facing computer scientists involve the development of efficient methods for rapidly creating swarm programs. For example, consider the situation in which a collection of swarm devices initially begin in a tightly formed group and then disperse or spread out over a particular environment. This dispersion behavior is illustrated below in Figure 2.

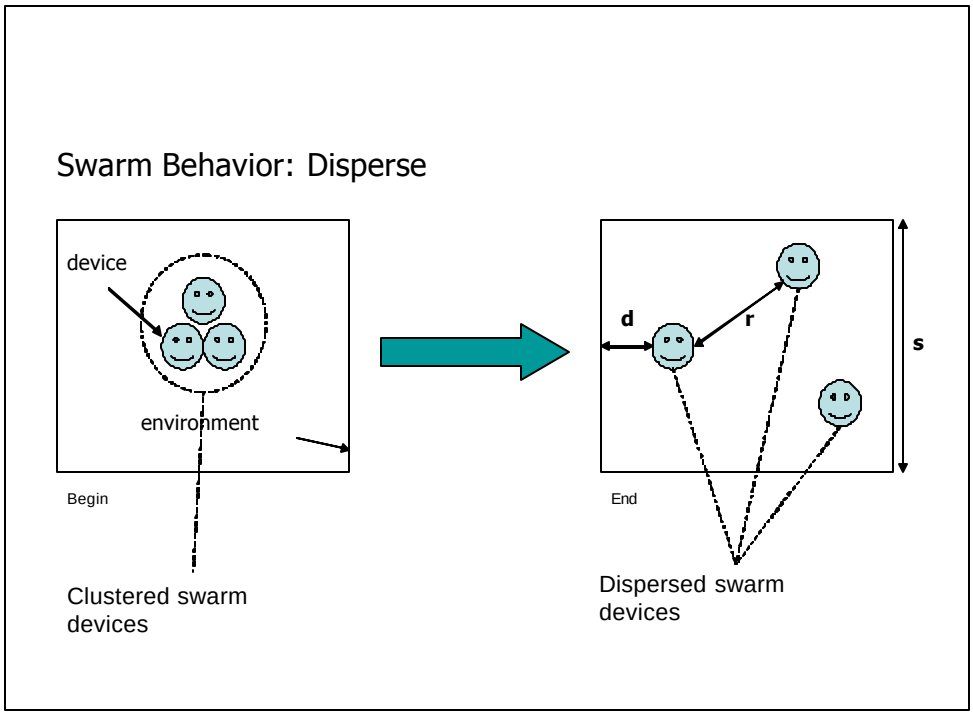

Figure 2. <u>The Disperse Behavior:</u> The devices begin in a clustered state and move randomly to spread apart some arbitrary distance r from one another and some distance d away from the environment perimeter.

Now suppose the behavioral requirements have changed to perform a more complex action. Assume that the swarm must now locate and rescue a particular object in the environment. Since the situation changed, it requires the swarm programmer to implement a new swarm program to provide the new desired swarm behavior. Hence the problem: swarm programs are specified by manually programming the desired behavior for each new situation. However, situations are constantly changing and new behaviors are always desired, which makes swarm programming a very time consuming and inefficient process.

In order to improve the efficiency associated with swarm programming, this project implemented a software system that allows swarm developers to create swarm programs at a much higher level of design.

3. A System to Facilitate Swarm Development

This project designed and implemented a software system that generated an acceptable swarm program for a given description of a high-level swarm behavior. It successfully accomplished the following objectives:

- Produced a software system to generate an acceptable swarm program from a high-level description of a swarm's behavior
  - Specified the high-level behavioral description as a combination of disperse, converge, and rescue, yielding the desired behavior of search and rescue
  - Developed the disperse, converge, and rescue behaviors for the high-level behavior descriptions
- Verified and Validated the functionality of the software system
  - Used a display simulator to interpret the behavior of the new swarm program

The result of this project provides a basis for developing swarm programs more efficiently. Instead of manually implementing every new swarm program, this project automates the programming process by generating swarm programs from a set of high-level swarm descriptions. This increases the efficiency of swarm programmers by transferring much of the development burden from the programmer to the actual program. The completion of this project has contributed to the infrastructure necessary to facilitate future advancements in swarm programming.

4. DOCUMENT OVERVIEW

The remaining chapters of this report focus on the underlying principles, design, and implementation of the software system, as well as the test procedure, results, and conclusion. Chapter 2 provides an overview of the software system and an introduction to the swarm behaviors pertinent to synthesizing a swarm program. Chapter 3 focuses on the design and implementation details of the software system in addition to the use of C++ in its development. The simulation methods used to test the software system, along with the results, are given in Chapter 4. In conclusion, the report summarizes and interprets the results, and makes recommendations to facilitate the future development of this project.

CHAPTER 2: SYNTHESIZING A SWARM PROGRAM

The principles for synthesizing a swarm program require a software system that is capable of fusing multiple swarm behaviors into a single new swarm behavior, and from that, synthesizing an acceptable swarm program. The essence of the system is to automate the process that a swarm programmer would perform in order to implement a new swarm program. This chapter provides an overview of the software system, as well as an introduction to swarm behaviors and their application to this project.

1. AN OVERVIEW OF THE SOFTWARE SYSTEM

The software system created in this project serves as an interface for swarm programmers, allowing them to create new swarm programs by simply selecting a set of swarm behaviors from an already existing code library. The code library currently possesses a limited number of simple swarm behaviors. The idea is to combine behaviors from this library to create a high-level description of the desired swarm behavior. The system will then combine the behaviors in the appropriate order to generate an acceptable swarm program. The model for this system is presented in Figure 3 on the next page.

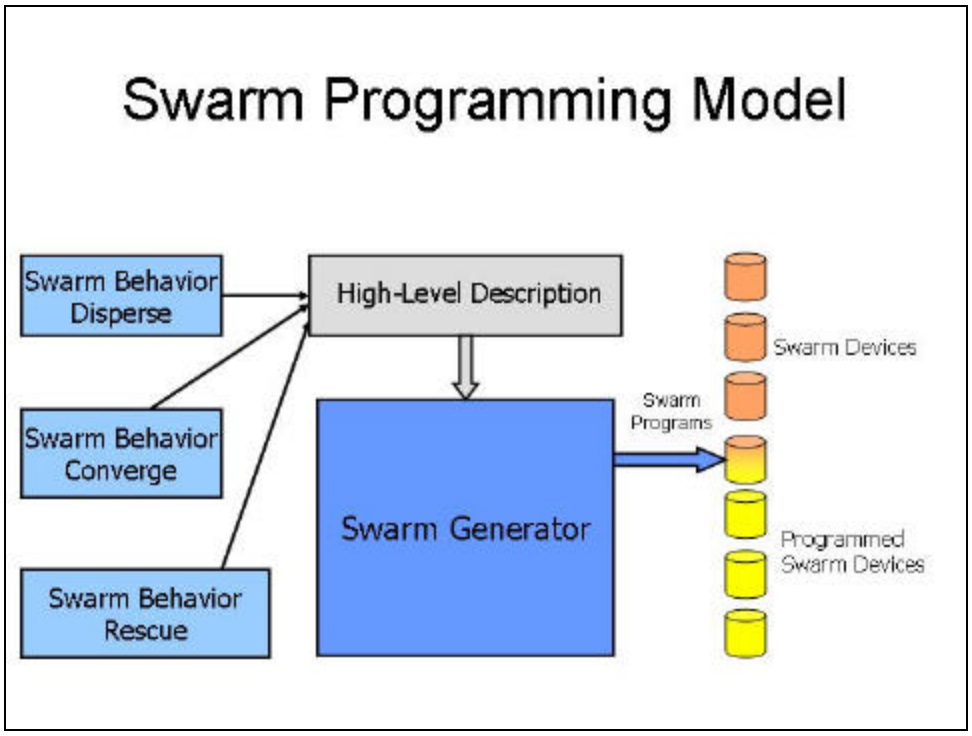

Figure 3. The Software System Model: The disperse, converge, and rescue behaviors are selected to form a high-level description, which is used by the Swarm Program Synthesizer to generate an acceptable swarm program for each device.

As mentioned previously, the software system accepts a high-level description of the desired swarm behavior as input. This description contains the source code for each of the selected swarm behaviors. A second input file, in addition to the high-level behavioral description, is also required. This second input file, is a text file, and serves as a template for the generation of the new swarm behavior. Once the code is generated successfully, it is compiled and output as a new swarm program for each device.

2. SWARM BEHAVIORS

A swarm behavior is a description of the global behavior that emerges from the collective interactions of individual swarm devices. Consider a colony of ants that are building a new anthill. If each ant performs the same individual behavior by piling its grain of sand in a specific location, then the swarm behavior that emerges is the creation

of an anthill. However, if the ant colony misbehaves, and each ant drops its grain of sand in some random location, no clear swarm behavior emerges. Thus, a large enough collection of devices exhibiting the same behavior dictates the behavior of the swarm.

Let us consider some of the swarm behaviors as they pertain to this project. The example application for this project was to synthesize a search-and-rescue swarm behavior. The idea is to search for a particular object, and then, once that object is found, the swarm should converge to its location and rescue it. The first, and one of the simplest behaviors that will be used, is called disperse. Disperse provides a method for a clustered collection of swarm devices to spread out over an existing area. Its objective is to achieve some acceptable distribution of devices over a given area, which will serve as an adequate method for searching the environment. While the devices are dispersing over the environment, it is beneficial to have some method for moving toward the object once it is found. The converge swarm behavior does just that; it enables swarm devices to scan for a particular object and close in on its position. The last behavior is rescue, which causes a swarm device to rescue a particular target once it is within range. The proper combination of these three behaviors into a single swarm program will serve as the search-and-rescue example application in the design and implementation of this project.

CHAPTER 3: DESIGN AND IMPLEMENTATION

This chapter presents the programming details for synthesizing a swarm program from a set of high-level behaviors. It also provides a discussion of the object-oriented design methodology and its application to swarm programming, as well as a class-by-class explanation of the software system created to generate an acceptable swarm program.

1. OBJECT-ORIENTED SWARM PROGRAMMING

In swarm programming, a swarm is viewed as a collection of devices where each device is represented by an object. In the C++ code for this project, the device object is defined by a class, which contains the necessary data and operations to describe the device. For example, in an ant colony, a class that describes an ant as an object might include information about the ant's color, vision, hearing, and speed. Hence, the class describes the properties that define an ant and provides operations, or member functions, that act on those properties to define a behavior. Each device object maintains its own state and behavior and is instantiated from a class definition. The program is then defined by an instantiated collection of these interacting device objects. This relationship is seen more clearly in Figure 4 presented on the next page.

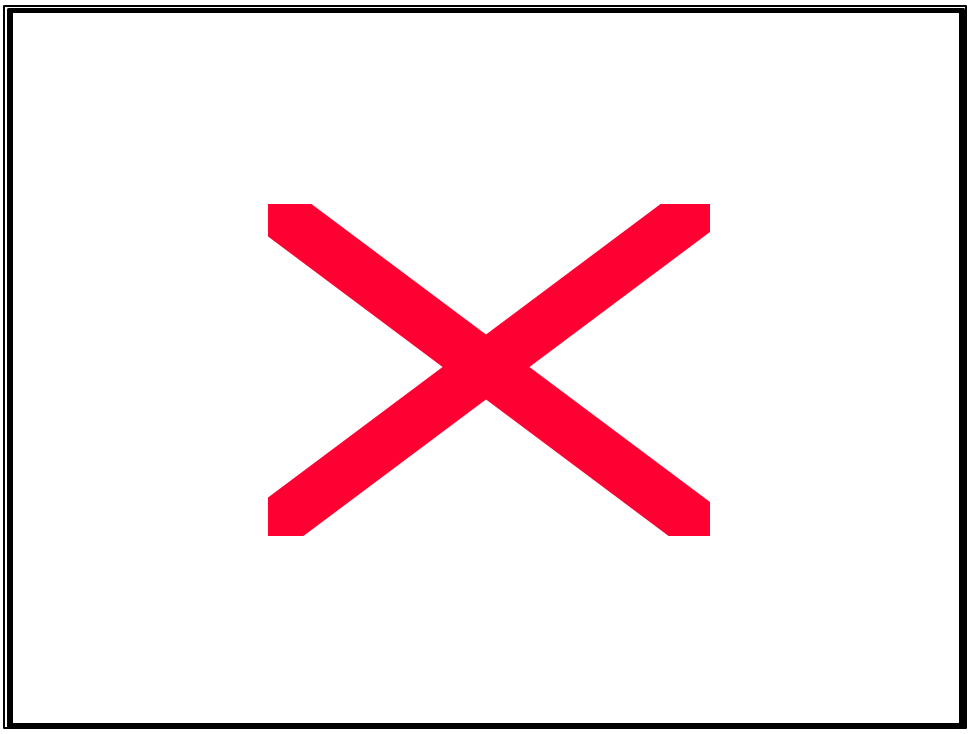

Figure 4. An Object's State and Behavior: The state and behavior is maintained within the object. The objects interact with one another in a program. http://www.santafe.edu/projects/swarm/swarmfest99-tutorial/

2. THE SWARM FRAMEWORK

The Swarm Programming Project here at the University of Virginia uses the Raptor Simulator for interpreting swarm programs. The Raptor Simulator is a general network simulator capable of simulating the dynamic network environments associated with swarm programming. The development of this thesis project was based on the concurrent development of the swarm framework that interfaces with the Raptor Simulator. The swarm framework is essentially a group of C++ classes that provide a model for a swarm's devices, behavior, and environment, as well as a method for viewing the swarm simulation. Development in this framework relies heavily on inheritance and information hiding OO design principles. For example, in this framework, the swarm programmer specifies a new device by first inheriting the properties of a basic device class and then

implementing the remaining details specific to the new device. Once the swarm program has been completed and compiles successfully, it is executed and its output is streamed to the framework's display component. The display interprets the output from the swarm program and shows the interactions of swarm devices with the surrounding environment. This framework served as the platform for the development of this thesis project.

3. THE SWARM GENERATOR

The purpose of this software system is to synthesize an acceptable swarm program from a high-level description of swarm behaviors. Specifying swarm behaviors at a high-level means selecting the name of the class behaviors that will be input into the software system. This project synthesized a search-and-rescue behavior using three behaviors: disperse, converge, and rescue. Each class behavior was created by inheriting its class from the base class `BasicAgent`. The class `BasicAgent` is included in the swarm framework to provide the fundamental structure for any inherited device or agent class. Every `BasicAgent`, or any class derived from it, contains a set of member functions that serve as a basis for defining the agent's behavior. The majority of a derived agent's behavior is implemented by specifying how a message is interpreted and what action is performed. This is done by implementing the `basic_agent_received_transmission()` and the `basic_agent_post_action()` functions, respectively. The code for the `BasicAgent` class along with other relevant code from the swarm framework is presented in Appendix F.

A discussion of the disperse, converge, and rescue behaviors was given in Chapter 2. This section describes the implementation of each behavior.

Disperse

The derived class, DisperseAgent, defines a simple method for swarm devices to move away from one another. The dispersion algorithm for this class is based completely on random movement. The class maintains the following two data members:

- **m_dispersing** – This value is either true or false and signifies if the agent is dispersing.
- **m_last_direction** – This variable specifies the (x, y) coordinates for a relative position

The `basic_agent_post_action()` function is overloaded to create a random relative position and move in that direction. Since devices cannot move into an already occupied area, another position is randomly chosen until an unoccupied area is found. Once an acceptable location is found, the device moves into this new position. This makes for an extremely simple, however, acceptable `DisperseAgent` class. The code for the `DisperseAgent` class is provided for review in Appendix E.

Converge

The specification of the `ConvergeAgent` class requires message processing and some notion of relative direction. Thus, the `ConvergeAgent` class maintains the following member variables:

- **m_converging** – This value is either true or false and signifies if the agent is moving in the correct direction.
- **m_power_current_msg** – This is a floating-point number that indicates the signal strength of the most recently received message.
- **m_power_of_last_received_msg** – This is a floating-point number that indicates the signal strength of the previously received message.

- **m_last_direction** – This variable specifies the (x, y) coordinates for a relative position

The `basic_agent_received_transmission()` function is overloaded to process messages by determining if the signal strength of the newly received message is stronger than that of the previous one. If the new signal is stronger, meaning `m_power_current_msg` is greater than `m_power_of_last_received_msg`, then the device is moving closer to the signal source, or converging on its target. The convergence algorithm specifies that the device move in the same direction if the signal gets stronger; if the single gets weaker, the device must adjust its position. This behavior is defined in the `basic_agent_post_action()` function. The code for the `ConvergeAgent` class is provided for review in Appendix E.

Rescue

The class definition for `RescueAgent` is relatively simple and does not really rescue anything at all. The behavior for rescue actually makes the device objects stop moving once they are within specific range of their target. This is done by overloading the `basic_agent_received_transmission()` function to determine whether the signal strength of a message was strong enough for the target to be considered within rescue range. If the target is within range, the device stops, which for our purposes means it is rescuing the target. This is accomplished by setting the single member variable, `m_rescuing`, to true if the signal strength of the received message crosses a certain threshold. The code for the `RescueAgent` class is provided for review in Appendix E.

Generating Swarm Programs

After each class behavior was created, the software system to generate a new synthesized behavior was developed. In general, the software system, or swarm generator, accepts any number of swarm behaviors as input, combines them in some logical order, and generates a new swarm program for each swarm device as output. The swarm generator is based on file processing and includes the `iostream` and `fstream` C++ class libraries. The `iostream` class is defined for standard input and output operations, whereas `fstream` is a file class, derived from `iostream`, for both reading and writing operations. In the example application for this project, the swarm generator accepts three behavioral class definitions – disperse, converge, and rescue –as input. Additionally, it requires two constant text files, `HeaderConstruct.txt` and `ClassConstruct.txt`, to serve as templates for the class structure of the new behavior. The behavior classes are then combined by selectively inserting code segments into the appropriate locations of the templates. Once the new class behavior, `NewAgentBehaviorAgent`, is generated, it is compiled into a swarm program, which is then tested for correctness and acceptability.

The `NewBehaviorAgent` class definition is a logical compilation of the `DisperseAgent`, `ConvergeAgent`, and `RescueAgent` class files that produces a swarm program with the desired search-and-rescue behavior. This process is illustrated by Figure 5 on the next page. The procedure for combining the three behaviors is controlled in the `main.cpp` file of the swarm generator project. In this file, a swarm generator object, called generator, is instantiated. The generator object invokes its class member functions to build the `NewBehaviorAgent`. The code for this file is given in `main.cpp` presented in Appendix B.

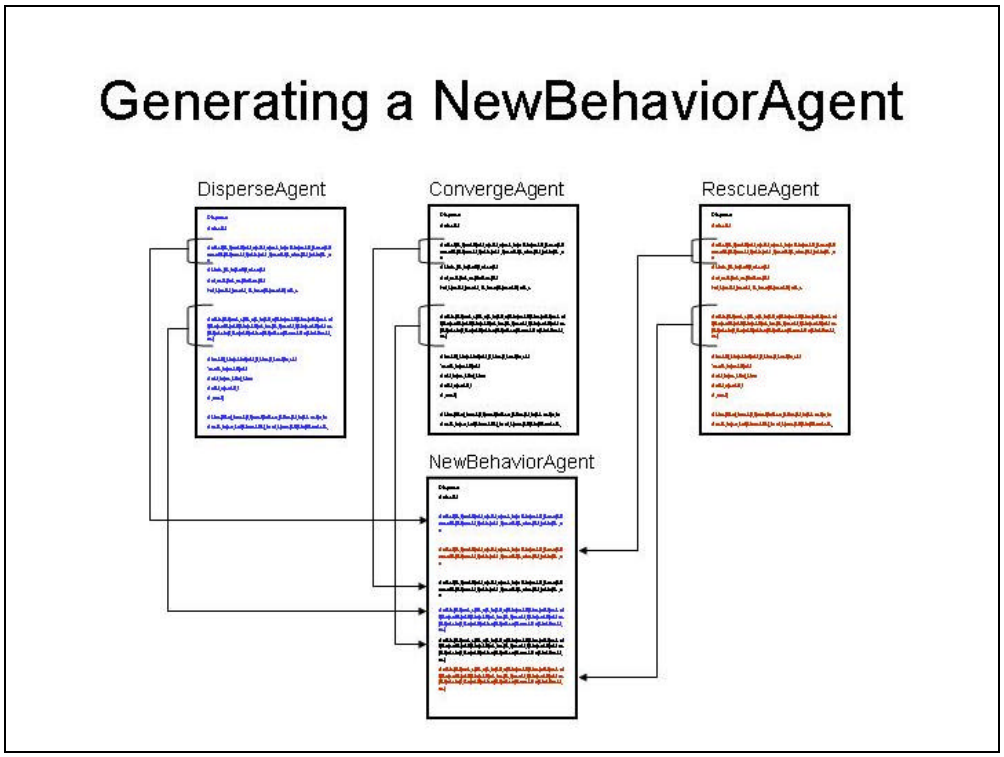

Figure 5. Generating a NewBehaviorAgent: The NewBehaviorAgent class definition is created by combining code segments from the DisperseAgent, ConvergeAgent, and RescueAgent class files in some logical order.

The Swarm Generator Class

The swarm generator object in `main.cpp` was instantiated from the `SwarmGenerator` class definition, which is the major component and focus of this project. The `SwarmGenerator` class provides file input and output operations specific to the swarm framework.

Upon execution of the program, the user is prompted to enter the names of the swarm behaviors files that will be combined in the program. The `SwarmGenerator` class provides this functionality with the `prompt_for_behavior_files()` member function. The function collects the behavior file names from the user and passes them to another function to be validated and stored within the program. If an incorrect filename is entered, the user is informed which filename is invalid and the program is terminated.

Once the filenames are valid and stored, the new behavior's class file construction may begin.

This procedure begins with a call to the `combine_cpp_function()`, which combines the contents of some specified function across the class definition files (.cpp file) of all behaviors. The `combine_cpp_function()` method assumes that each behavior has its own implementation of the specified function. This is an acceptable assumption because each swarm behavior was derived from the same base class and thus, contains a set of pure virtual functions, which requires each derived class to include its own version of all of those function definitions. After the `combine_cpp_function()` is invoked for each member function in the derived class definitions, the `end_combine_cpp()` routine is called, which completes file construction process for the new behavior's class definition.

A similar procedure is used to process the class header files, which maintains the same assumptions about file structure made earlier. First, the `combine_header_file()` is called, which combines portions of the class header files based on the start and end parameters passed to the function. Then, by invoking the `end_combine_h()` function, the process for combing the header file is completed.

Delving further into the implementation of the `SwarmGenerator` class reveals a few other important member functions. The main functionality of this class deals primarily with seeking to a particular point in a file and copying the code from that file to a particular point in another file.

The `seek_function_content()` member allows the swarm generator to prepare a file for processing at a particular location. Data from a specified point in the input file is copied to the appropriate point in the output file using the `copy_cpp_function_content()`

and `copy_header_file_content()` functions. Most of the swarm generator's complexity exists in these two files.

The `copy_cpp_function_content()` first calls the `seek_function_content()` member to prepare the input file for processing. Then, once the input file pointer is positioned in the proper location, the content of the function is copied. This member function uses the code block structure of a C++ function (function definitions are enclosed by open and close curly braces) to determine when the end of the function has been reached and, thus, when it can stop copying the file. The algorithm for this procedure is simple; it maintains separate counts for all open and close curly braces encountered while processing the function, and, when the number of open curly braces equals the number of close curly braces, the end of the function has been reached.

However, there is a problem because not every line of the input file's function should always be copied to the output file. The reason is that the output file's function may already contain that particular line, in which case a duplicate will cause an error in the final program. Thus, the `copy_cpp_function_content()` restricts the insertion of duplicate lines by maintaining what lines already exist in the output file's function and then comparing those lines to the potential line coming from the input file. If the potential line already exists in the output file, or it is not inserted into a separate code block, then that line is discarded and the next line is processed.

The implementations of the `copy_cpp_function_content()` and `copy_header_file_content()` are very similar. The only major difference is that the `copy_header_file_content()` takes an additional parameter to determine which parts of the header file to copy. It copies the contents of the header input file – delimited by the start and end

parameters – into the proper location of the output file. Once again, this function restricts duplications for the same reason presented earlier.

The code for the `SwarmGenerator` class discussed in this section is presented in Appendix A. The comments located in these class files provide an explanation for every function and explain some of the more program specific implementation details. After the swarm generator class and the example search-and-rescue application were constructed, simulation methods were used to test the results.

CHAPTER 4: TESTING

The software system was tested using swarm simulation methods based on the Raptor Simulator developed at the University of Virginia. The goal of the test was to combine three high-level swarm behaviors, disperse, converge, and rescue, in order to create an acceptable swarm program that exemplified a new search-and-rescue behavior. The software system was successful in synthesizing an acceptable swarm program from high-level swarm behaviors. This chapter discusses the simulation methods used to test the results, as well as the overall performance of the software system

1. SWARM SIMULATION METHODS

The correctness of the software system was tested primarily on two levels. First, did the behavior generated by the software system compile? Secondly, if the behavior compiled successfully, was it an acceptable swarm program? The first question or level of testing requires a simple yes or no answer. However, the second question requires some interpretation of the swarm program. The swarm program's acceptability is interpreted visually, using the display component of the swarm framework. The display component is a C++ implementation that uses the OpenGL and GLUT code libraries. It interprets the output of a swarm program and creates a visual representation of the behavior. The small grey spheres represent the swarm devices, and the window frame encapsulates the swarm environment. The display component gives a simple, yet effective method for visually interpreting a swarm program.

The interpretation of a swarm program is based on the fact that the behavior of the swarm emerges from the collective behaviors of the individual devices. As mentioned earlier, this principle refers to the idea that the swarm behavior must be resilient to the

misbehavior of a few individual devices. However, the question is then, how many devices are allowed to misbehave before the behavior is no longer acceptable? The answer is simple: a behavior is deemed unacceptable once it is no longer clear what behavior was originally intended.

2. PERFORMANCE

The software system was successful in synthesizing a swarm program on both levels of testing. The original question and focus of this project was:

Given a high-level description of a swarm's behavior can we generate an acceptable swarm program?

The answer is yes, the software system succeeded in generating an acceptable swarm program from the given high-level specification. During the simulation of the swarm program, the search-and-rescue behavior emerged, showing the devices disperse until the target was located, converged on, and then rescued.

The major strengths of the software system are that it proves that the synthesis of swarm programs from high-level programs is possible and its implementation details are fairly simple and easy to understand. Simple implementation methods make the system's functionality easily extendible, which will facilitate future swarm development.

However, a weakness of the software system is that even though it generated an acceptable swarm program it is not flexible enough to deal with the many ways to specify the same swarm program. In other words, the software system does not handle all implementations of the same swarm program equally well.

CHAPTER 5: CONCLUSIONS

1. SUMMARY

This project developed a software system for synthesizing swarm programs and proved that the generation of an acceptable swarm program from a set of high-level swarm behaviors is possible. The system, or swarm generator, produced a search-and-rescue program from the disperse, converge, and rescue behaviors that exhibited the same behavior as the swarm program developed manually by the programmer.

2. INTERPRETATION

The significance of generating a swarm program that behaved the same as the manually programmed version is that it proves the automation of the swarm programming process is certainly possible. However, there are some important points to consider when interpreting the results of this project. The first issue deals with the structure of the behaviors combined in the search-and-rescue test application. The disperse, converge, and rescue behaviors all share a common class file structure due to the fact that each behavior was derived from the same base class. This means that the member functions in the three class behaviors all have the same function names. For example, each of the three swarm behaviors contains a function with the name, `basic_agent_start()`. The swarm generator is heavily dependent on this property, somewhat limiting its ability to combine behaviors derived from different base classes.

Another issue affecting the performance of the swarm generator follows from a programmer's ability to implement the same program in many different ways. This means that variations in the implementation of a swarm behavior may produce functional variations in the swarm program generated by the software system. Although in some

cases these issues may hinder the generation of an acceptable swarm program, the software system provides methods for accommodating the changes required by a swarm programmer to circumvent these obstacles. Hence, the current weaknesses in the software system are not major limitations and provide an excellent entry point for additional research on this project.

3. RECOMMENDATIONS

This project provides a strong basis for which future research can begin to build. The intent of the recommendations given in this section is to create a spiral development process to increase the functionality of the software system through the testing and modifications provided by future projects.

The recommendations for future researchers are to analyze the limitations of the software system and expand its current functionality to be more flexible. The researcher should develop a test harness, including multiple applications, to explore in detail the limitations of the software system. For instance, at the time of this project, the swarm framework provided only one base class, `BasicAgent`, from which all other swarm behaviors were derived. Thus, the software system was not tested with behaviors derived from different base classes. It would be beneficial to expand the swarm framework to include multiple base classes and then develop a test application to generate a swarm program from a set of behaviors inherited from a set of different base classes.

Another area of this project that needs to be investigated is the software system's ability to handle multiple implementations of the same swarm behaviors. The disperse, converge, and rescue behaviors are very basic and were developed as an example application to prove that the generation of a swarm program from a set of high-level

behaviors was feasible. Researchers should enhance the functionality of these behaviors and then test their synthesis using the swarm generator. A comparison of the test results with the results of this project should provide valuable information regarding the current limitations of the software system. Based on these findings, the researcher should make the appropriate modifications to the swarm generator, increasing its overall functionality.

BIBLIOGRAPHY

[1] American Philosophical Society "The first stored program for a computer." Retrieved September 2001 from <http://www.amphilsoc.org/library/exhibits/treasures/vonneuma.htm>

[2] Barber, K. and Martin, C. (2001, June). "Dynamic Reorganization of Decision-Making Groups." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[3] Barber, K., McKay, r., MacMahon, M., Martin, C., Lam, D., Goel, A., Han, D., and Kim, J. (2001, June). "Sensible Agents: An Implemented Multi-Agent System and Testbed." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[4] Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999, July). Swarm Intelligence: From Natural to Artificial Systems. :Oxford University Press

[5] Coelho, A., Weingaertner, D., and Gomide, F. (2001, June). "Evolving Coordination Strategies in Simulated Robot Soccer." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[6] DeLoach, S. (2001, June). "Specifying Agent Behavior as Concurrent Tasks." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[7] Evans, D. (2000, July). "Programming the Swarm." Retrieved September, 2001 from <http://www.cs.virginia.edu/~evans/swarm/nsf-proposal.pdf>

[8] Evans, D. (2000, July). "Swarm Programming: How to program a Micronet" Retrieved October, 2001<http://www.cs.virginia.edu/~evans/talks/index.html>

[9] Fatima, S. and Wooldridge, M. (2001, June). "Adaptive Task and Resource Allocation in Multi-Agent Systems." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[10] Hodges, A. (2001) "The Alan Turing Internet Scrapbook." Retrieved September 2001 from <http://www.turing.org.uk/turing/scrapbook/computer.html>

[11] Horling, B., Benyo, B. and Lesser, V. (2001, June). "Using Self-Diagnosis to Adapt Organizational Structures." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[12] Jim, K. and Giles, C. (2001, June). "How Communication Can Improve the Performance of Multi-Agent Systems" International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[13] Kalin, M. (1999). Applications Programming in C++. Upper Saddle River, New Jersey: Prentice Hall

[14] Kennedy, J. and Eberhart R. (2001). Swarm Intelligence. San Diego, CA: Academic Press.

[15] Koeing, S. and Yaxin, L. (2001, June). "Terrain Coverage with Ant Robots: A Simulation Study." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[16] Lawlor, S. C. (1998). The Art of Programming: Computer Science with C++. 20 Park Plaza, Boston, Massachusetts: PWS Publishing Company

[17] Lita, L., Jamieson, S., and Thrun, S. (2001, June). "A System for Multi-Agent Coordination in Uncertain Environments." International Conference on Autonomous Agents. Proc. of the Fifth International Conf. on Autonomous agents, May 28-June 1 2001, Montreal, Quebec, Canada. Association for Computing Machinery

[18] Müller, P., Introduction to Object-Oriented Programming Using C++ (1997) Retrieved January 2002 from <http://www.zib.de/Visual/people/mueller/Course/Tutorial/tutorial.html>

[19] Multi-agent World, MAAMAW '94, Odense, Denmark, August 3-5, 1994. Berlin Heidelberg: Springer-Verlag

[20] Schildt, H. (1998). The Complete Reference: C++ Third Edition. Berkley, California: Osborne McGraw-Hill

[21] The Swarm Development Group "The Swarm Development Group." Retrieved December 2001 from http://www.swarm.org

APPENDIX A: THE SWARM GENERATOR CLASS FILES

```
/*=============================================================*/
/*                      SWARMGENERATOR.H                       */
/*  ---------------------------------------------------------  */
/*  This class uses the C++ file I/O libraries to combine      */
/*  multiple class files specific to the structure of a swarm  */
/*  program.                                                   */
/*  ---------------------------------------------------------  */
/*  The following resource files should accompany the use of   */
/*  the SwarmGenerator class.                                  */
/*     ClassConstruct.txt  -- Template for swarm .cpp file     */
/*     HeaderConstruct.txt -- Template for swarm .h file       */
/*     LowPowerNewBehavior.cpp -- see comments in file         */
/*     LowPowerNewBehavior.h      -- see comments in file      */
/*  ---------------------------------------------------------  */
/*  Author: Errol McEachron                 Date: 03/20/2002  */
/*  ---------------------------------------------------------  */
/*=============================================================*/

#ifndef SWARM_GENERATOR_H
#define SWARM_GENERATOR_H

#include <iostream>
#include <string>
#include <fstream>
#include <sstream>


using namespace std;

class SwarmGenerator
{
public:
       SwarmGenerator(int iNumBehaviors);
       ~SwarmGenerator();

       void prompt_for_behavior_files();

       void combine_cpp_function(char* strFunctionName);
```

```
	void combine_header_file(const char *strStart, const char *strEnd);

	void end_combine_cpp();
	void end_combine_h();

protected:
	bool function_is_allowed(char *function);
	bool duplicate_is_allowed(char *duplicate);
	bool substr(char* source, const char* substr);

	void set_behavior_names(string strName, int index);
	void set_behavior_cpp_filenames(string strFilename, int index);
	void set_behavior_header_filenames(string strFilename, int index);
	void set_behavior_function_names(char* strFunctionName);

	void seek_function_content(fstream &fin, const char *strDelimiter);

	void copy_cpp_file_until(fstream &fin, ofstream &fout, const char *strDelimiter);
	void copy_header_file_until(fstream &fin, ofstream &fout, const char *strDelimiter);
	void copy_cpp_function_content(fstream &fin, ofstream &fout, const char *strFunctionName);
	void copy_header_file_content(fstream &fin, ofstream &fout,const char* strStart, const char *strEnd);

private:
	SwarmGenerator(const SwarmGenerator &swarm_generator);

	int m_number_of_behaviors;

	char **m_behavior_name;
	char **m_behavior_filename_cpp;
	char **m_behavior_filename_h;
	char **m_behavior_function_name;

	fstream **m_fin_behavior_cpp;
	fstream **m_fin_behavior_h;
	fstream m_fin_construct_cpp;
	fstream m_fin_construct_h;

	ofstream m_fout_behavior_cpp;
	ofstream m_fout_behavior_h;

	stringstream m_sstream_container_cpp;
	stringstream m_sstream_container_h;
```

```
};
#endif
```

```
/*=============================================================*/
/*                    SWARMGENERATOR.CPP                       */
/*  -----------------------------------------------------------  */
/*  This class uses the C++ file I/O libraries to combine      */
/*  multiple class files specific to the structure of a swarm  */
/*  program.                                                   */
/*  -----------------------------------------------------------  */
/*  The following resource files should accompany the use of   */
/*  the SwarmGenerator class.                                  */
/*     ClassConstruct.txt  -- Template for swarm .cpp file     */
/*     HeaderConstruct.txt -- Template for swarm .h file       */
/*     LowPowerNewBehavior.cpp -- see comments in file         */
/*     LowPowerNewBehavior.h      -- see comments in file      */
/*  -----------------------------------------------------------  */
/*  Author: Errol McEachron                 Date: 03/20/2002  */
/*  -----------------------------------------------------------  */
/*=============================================================*/

#include "SwarmGenerator.h"

// SwarmGenerator(): Constructor
SwarmGenerator::SwarmGenerator(int iNumBehaviors)
       : m_number_of_behaviors(iNumBehaviors)
{
       m_behavior_name = new char*[m_number_of_behaviors];
       m_behavior_filename_cpp = new char*[m_number_of_behaviors];
       m_behavior_filename_h = new char*[m_number_of_behaviors];
       m_behavior_function_name = new char*[m_number_of_behaviors];

       m_fin_behavior_cpp = new fstream*[m_number_of_behaviors];
       m_fin_behavior_h = new fstream*[m_number_of_behaviors];

       m_fin_construct_cpp.open("ClassConstruct.txt", ios::in);
       m_fin_construct_h.open("HeaderConstruct.txt", ios::in);

       m_fout_behavior_cpp.open("NewBehaviorAgent.cpp", ios::trunc | ios::in);
       m_fout_behavior_h.open("NewBehaviorAgent.h", ios::trunc | ios::in);
}
```

```
// prompt_for_behavior_files(): Prompts user for behavior class
// names and sets the corresponding filenames
void SwarmGenerator::prompt_for_behavior_files()
{
	cout << "Enter the filename of all desired behaviors" << endl;
	cout << "(excluding file extensions) beginning with" << endl;
	cout << "the default behavior and concluding with the" << endl;
	cout << "terminating behavior." << endl << endl;

	string strTemp;
	for (int i = 0; i < m_number_of_behaviors; i++)
	{
		cout << "Enter the name of behavior #" << i + 1 << endl;
		cin >> strTemp;

		set_behavior_names(strTemp, i);
		set_behavior_cpp_filenames(strTemp, i);
		set_behavior_header_filenames(strTemp, i);
	}

}

// combine_cpp_functions():  Controls the program flow for combining the // .cpp behavior files for a specific function
(strFunctionName)
// (Insertion points in the ClassConstruct.txt file are denoted by @)
void SwarmGenerator::combine_cpp_function(char* strFunctionName)
{
	set_behavior_function_names(strFunctionName);
	copy_cpp_file_until(m_fin_construct_cpp, m_fout_behavior_cpp, "@");
	for (int i = 0; i < m_number_of_behaviors; i++)
	{
		copy_cpp_function_content(*m_fin_behavior_cpp[i], m_fout_behavior_cpp, m_behavior_function_name[i]);
	}
}
```

```
// combine_header_files(): Controls the program flow for combining the // .h behavior files from strStart to strEnd
(Insertion points in the
// HeaderConstruct.txt file are denoted by @)
void SwarmGenerator::combine_header_file(const char *strStart, const char *strEnd)
{
	copy_header_file_until(m_fin_construct_h, m_fout_behavior_h, "@");
	for (int i = 0; i < m_number_of_behaviors; i++)
	{
		copy_header_file_content(*m_fin_behavior_h[i], m_fout_behavior_h, strStart, strEnd);
	}
}

// end_combine_cpp(): Completes the .cpp file copy process by copying
// the remaining portion of the ClassConstruct.txt template file
// ($ is used to denote the end of the ClassConstruct.txt file)
void SwarmGenerator::end_combine_cpp()
{
	copy_cpp_file_until(m_fin_construct_cpp, m_fout_behavior_cpp, "$");
}


// end_combine_h(): Completes the .h file copy process by copying
// the remaining portion of the ClassConstruct.txt template file
// ($ is used to denote the end of the ClassConstruct.txt file)
void SwarmGenerator::end_combine_h()
{
	copy_header_file_until(m_fin_construct_h, m_fout_behavior_h, "$");
}


// set_behavior_names(): Initializes m_behavior_name array to
// corresponding class behavior names
void SwarmGenerator::set_behavior_names(string strName, int index)
{
	m_behavior_name[index] = new char[strName.size() + 1];
	strcpy(m_behavior_name[index], strName.c_str());
}
```

```
// set_behavior_cpp_filenames(): Initializes m_behavior_filename_cpp
// array to corresponding class behavior names and initializes
// m_fin_behavior_cpp array file pointers to the corresponding filenames
void SwarmGenerator::set_behavior_cpp_filenames(string strFilename, int index)
{
            string strTemp = strFilename + ".cpp";
            m_behavior_filename_cpp[index] = new char[strTemp.size()+1];
            strcpy(m_behavior_filename_cpp[index], strTemp.c_str());

            m_fin_behavior_cpp[index] = new fstream(m_behavior_filename_cpp[index]);
            if(!m_fin_behavior_cpp[index]->is_open())
            {
                  printf("File %s is not found.\n", m_behavior_filename_cpp[index]);
                  exit(1);
            }
}


// set_behavior_header_filenames(): Initializes m_behavior_filename_h
// array to corresponding class behavior names and initializes
// m_fin_behavior_h array file pointers to the corresponding filenames
void SwarmGenerator::set_behavior_header_filenames(string strFilename, int index)
{
            string strTemp = strFilename + ".h";
            m_behavior_filename_h[index] = new char[strTemp.size() + 1];
            strcpy(m_behavior_filename_h[index], strTemp.c_str());

            m_fin_behavior_h[index] = new fstream(m_behavior_filename_h[index]);
            if(!m_fin_behavior_h[index]->is_open())
            {
                  printf("File %s is not found.\n", m_behavior_filename_h[index]);
                  exit(1);
            }
}
```

```
// set_behavior_function_names(): Prepends the behavior class names to
// strFunctionName and initializes m_behavior_function_name
void SwarmGenerator::set_behavior_function_names(char* strFunctionName)
{
	int strPos = 0;
	string strTemp;

	for (int i = 0; i < m_number_of_behaviors; i++)
	{
		strTemp = m_behavior_filename_cpp[i];
		strPos = strTemp.find(".cpp");
		strTemp.replace(strPos, 4, strFunctionName);
		m_behavior_function_name[i] = new char[strTemp.size() + 1];
		strcpy(m_behavior_function_name[i], strTemp.c_str());
	}
}


// seek_function_content(): Sets file pointer to first char immediately
// after strDelimiter
void SwarmGenerator::seek_function_content(fstream &fin, const char *strDelimiter)
{
	char strCompare[256];
	int offset = 0;

	do
	{
		fin.getline(strCompare,256);
	} while (!substr(strCompare, strDelimiter) && !fin.eof());

	offset = (int)fin.tellg();
	fin.seekg(offset, ios::beg);
}
```

```
// copy_cpp_file_until(): Copies contents of fin to fout until strDelimiter
// is reached in fin and stores copy of contents in m_sstream_container_cpp
// for duplicate checking later
void SwarmGenerator::copy_cpp_file_until(fstream &fin, ofstream &fout, const char *strDelimiter)
{
	char buffer[256];
	string line;
	stringstream temp_stream;

	fin.getline(buffer, 256);
	line = buffer;
	while ((strDelimiter != line) && !fin.eof())
	{
		temp_stream << line << endl;
		fout << line << endl;
		fin.getline(buffer, 256);
		line = buffer;
	}
	m_sstream_container_cpp << temp_stream.str() << endl;
}


// copy_header_file_until(): Copies contents of fin to fout until strDelimiter
// is reached in fin and stores copy of contents in m_sstream_container_h
// for duplicate checking later
void SwarmGenerator::copy_header_file_until(fstream &fin, ofstream &fout, const char *strDelimiter)
{
	char buffer[256];
	string line;
	stringstream temp_stream;

	fin.getline(buffer, 256);
	line = buffer;
	while ((strDelimiter != line) && !fin.eof())
	{
		temp_stream << line << endl;
		fout << line << endl;
		fin.getline(buffer, 256);
		line = buffer;
	}
	m_sstream_container_h << temp_stream.str() << endl;
}
```

```
// copy_cpp_function_content(): Copies the contents of fin (the .cpp member
// function specified by strFunctionName) to fout (the new .cpp file)
void SwarmGenerator::copy_cpp_function_content(fstream &fin,  ofstream &fout, const char *strFunctionName)
{
	// Positions fin's file pointer to the first byte
	// after the strFunctionName
	seek_function_content(fin, strFunctionName);

	// Is set to true each time a new code block (left curly
	// brace { is encountered), which allows functions with
	// the same name to be called in different code blocks
	bool new_code_block = false;

	int number_left_braces = 1;
	int number_right_braces = 0;

	char buffer[256];
	string line;

	char *temp_param1;
	char *temp_param2;

	// The first { of the function is skipped (removed from the fin stream)
	// Note that this is why number_left_braces is intialized to 1
	fin.ignore(1); // Skip first "{" because it is already in fout

	fin.getline(buffer, 256);
	line = buffer;

	while (number_left_braces != number_right_braces)
	{
		string temp_string = m_sstream_container_cpp.str();
		temp_param1 = (&temp_string[0]); // convert from strings
		temp_param2 = (&line[0]);          // to char*

		// Determine if the fin line can be inserted into fout
		if (new_code_block
			|| duplicate_is_allowed(temp_param2)
			|| !(substr(temp_param1, temp_param2)))
		{
```

```
                fout << line << endl;
                m_sstream_container_cpp << line << endl;
        }

        fin.getline(buffer, 256);
        line = buffer;

        // Keep track of code blocks
        if (substr(buffer, "{"))
        {
                new_code_block = true;
                number_left_braces++;
        }
        if (substr(buffer, "}"))
        {
                number_right_braces++;
        }

        // This sets the boolean value of the new behavior agent to
        // false once the end of the terminating behavior has been reached
        string temp_name = (m_behavior_name[m_number_of_behaviors-1]);
        temp_name += "::basic_agent_post_action";
        temp_param1 = (char *)(&strFunctionName[0]);
        temp_param2 = (char *)(&temp_name[0]);
        if ((number_right_braces == 1) &&
                (substr(temp_param1, temp_param2)))
        {
                fout << "m_performing_behavior = false;" << endl;
        }
    }
}
```

```
// copy_header_file_content(): Copies the contents of fin (the .h header file)
// to fout (the new .h header file) beginning at strStart and ending at strEnd
void SwarmGenerator::copy_header_file_content(fstream &fin,  ofstream &fout,const char *strStart, const char *strEnd)
{
	// Positions fin's fp to the first char after the strStart
	seek_function_content(fin, strStart);

	char buffer[256];
	string line;
	string strCompare = strEnd;

	char *temp_param1;
	char *temp_param2;

	fin.getline(buffer, 256);
	line = buffer;

	// Copies file until strEnd or the end of the fin file is reached
	while (!substr(buffer, strEnd) && !fin.eof())
	{
		string temp_string = m_sstream_container_h.str();
		temp_param1 = (&temp_string[0]);
		temp_param2 = (&line[0]);

		if (function_is_allowed(temp_param2)
			&& (duplicate_is_allowed(temp_param2)
			|| !(substr(temp_param1, temp_param2))))
		{
			fout << line << endl;
			m_sstream_container_h << line << endl;
		}

		fin.getline(buffer, 256);
		line = buffer;
	}
}
```

```
// substr(): returns true if substr is a substring of source
// else returns false (checks character by character)
bool SwarmGenerator::substr(char* source, const char* substr)
{
    char* pstr = source;
    char* psub = (char *)substr;
       char* pbtb = (char *)substr;

       int len_str = strlen(source);
    int len_sub = strlen(substr);

       for ( ;len_str >= len_sub;++pstr,--len_str)
       {
        char *p=pstr;

        // Compare characters in source to substr while
              // they are equal and substr not at end
              while (*p == *psub && *psub)
              {
            ++p;
            ++psub;
        }

              // If end of substr has been reached, substr must
              // exist in source so return true else try again
        if (!*psub)
              {
                    return true;
              }
              // reset psub back to beginning (pbtb)
              else
              {
                    psub = pbtb;
              }
    }
       // If substr is not a member of source return false
    return false;
}
```

```
// duplicate_is_allowed(): if duplicate is a member of
// the valid duplicate set return true else false
bool SwarmGenerator::duplicate_is_allowed(char *duplicate)
{
	char *valid_duplicate1 = "{";
	char *valid_duplicate2 = "}";

	if(substr(duplicate, valid_duplicate1)) return true;
	if(substr(duplicate, valid_duplicate2)) return true;
	return false;
}


// function_is_allowed(): if function is a member of the
// behavior name set return false else true
bool SwarmGenerator::function_is_allowed(char *function)
{
	for (int i = 0; i < m_number_of_behaviors; i++)
	{
		if(substr(function, m_behavior_name[i])) return false;
	}
	return true;
}


//~SwarmGenerator(): Destructor
SwarmGenerator::~SwarmGenerator()
{
/*-----------------------------------------------------------------*/
	m_fout_behavior_cpp.close();
	m_fin_construct_cpp.close();
	m_fout_behavior_h.close();
	m_fin_construct_h.close();

/*-----------------------------------------------------------------*/
	for (int i = 0; i < m_number_of_behaviors; i++)
	{
		if(m_fin_behavior_cpp[i] != NULL)
		{
			m_fin_behavior_cpp[i]->close();
		}
	}
```

```
	delete [] m_fin_behavior_cpp;

/*-----------------------------------------------------------------*/
	for (i = 0; i < m_number_of_behaviors; i++)
	{
		if(m_fin_behavior_h[i] != NULL)
		{
			m_fin_behavior_h[i]->close();
		}
	}

	delete [] m_fin_behavior_h;

/*-----------------------------------------------------------------*/
	for(i = 0; i < m_number_of_behaviors; i++)
	{
		delete [] m_behavior_function_name[i];
		m_behavior_function_name[i] = NULL;
	}

	delete [] m_behavior_function_name;

/*-----------------------------------------------------------------*/
	for(i = 0; i < m_number_of_behaviors; i++)
	{
		delete [] m_behavior_filename_cpp[i];
		m_behavior_filename_cpp[i] = NULL;
	}

	delete [] m_behavior_filename_cpp;

/*-----------------------------------------------------------------*/
	for(i = 0; i < m_number_of_behaviors; i++)
	{
		delete [] m_behavior_filename_h[i];
		m_behavior_filename_h[i] = NULL;
	}

	delete [] m_behavior_filename_h;

/*-----------------------------------------------------------------*/
```

```
}
```

APPENDIX B: MAIN.CPP

```
/*=============================================================*/
/*                          MAIN.CPP                           */
/*  ---------------------------------------------------------  */
/*  This file uses the Swarm Generator class to build a swarm  */
/*  program from three seperate swarm entity classes.  The     */
/*  specific classes used in this example are DisperseAgent,   */
/*  ConvergeAgent, and RescueAgent.                            */
/*=============================================================*/

#include "SwarmGenerator.h"
#include <iostream>
#include <string>

using namespace std;

int main()
{
      // The SwarmGenerator class constructor requires an integer
      // specifying the number of behavior files that will be
      // combined.
      int numBehaviors = 0;
      cout << "Enter the number of behaviors \n";
      cin >> numBehaviors;

      SwarmGenerator* generator = new SwarmGenerator(numBehaviors);

      // Prompt user for behavior class names (exluding file extensions)
      generator->prompt_for_behavior_files();

      // Generate the NewBehaviorAgent.cpp file by copying the functions
      // of the corresponding classes (this process makes use of
      // ClassConstruct.txt -- a .txt file, which serves as a template
      generator->combine_cpp_function("::basic_agent_start");
      generator->combine_cpp_function("::basic_agent_pre_action");
      generator->combine_cpp_function("::basic_agent_received_transmission");
```

```
      generator->combine_cpp_function("::basic_agent_answer_query");
      generator->combine_cpp_function("::basic_agent_post_action");
      generator->combine_cpp_function("::basic_agent_stop");
      generator->end_combine_cpp();

      // Generate the NewBehaviorAgent.h file by copying the data members
      // of the corresponding class header files
      generator->combine_header_file("private:", "};");
      generator->end_combine_h();

      generator->~SwarmGenerator();

      return 0;
}
```

APPENDIX C: THE TEMPLATE CONSTRUCT FILES

```
/*=============================================================*/
/*                     HEADERCONSTRUCT.TXT                     */
/*  ---------------------------------------------------------  */
/*  This text file is used by the swarm generator to guide the */
/*  construction of the swarm program.  The structure of the   */
/*  template file is based of the structure of the base class  */
/*  BasicAgent.h                                               */
/*=============================================================*/

#ifndef NEW_BEHAVIOR_AGENT_H
#define NEW_BEHAVIOR_AGENT_H

#include "BasicAgent.h"
#include "Position.h"

class NewBehaviorAgent : public virtual BasicAgent
{
public:
       NewBehaviorAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity);

protected:
       void basic_agent_start();
       void basic_agent_pre_action();
       void basic_agent_received_transmission(ReceivedTransmissionMessage *msg);
       void basic_agent_answer_query(int observer_addr, QueryMessage *msg);
       void basic_agent_post_action();
       void basic_agent_stop();

private:
       NewBehaviorAgent(const NewBehaviorAgent &New_Behavior_agent);

       const unsigned int m_random_seed;
       bool m_performing_behavior;
@
};
#endif
$
```

```
/*============================================================*/
/*                      CLASSCONSTRUCT.TXT                    */
/*  ----------------------------------------------------------  */
/*  This text file is used by the swarm generator to guide the */
/*  construction of the swarm program.  The structure of the   */
/*  template file is based of the structure of the base class  */
/*  BasicAgent.cpp                                             */
/*============================================================*/

#include "NewBehaviorAgent.h"
#include "Debug.h"


NewBehaviorAgent::NewBehaviorAgent(int my_addr, int world_addr, const Gate &world_gate,
                                                    double sensor_sensitivity)
      : BasicAgent(my_addr, world_addr, world_gate, sensor_sensitivity), m_random_seed(rand())
{

}


void NewBehaviorAgent::basic_agent_start()
{
      srand(m_random_seed);
      m_performing_behavior = true;
@
}



void NewBehaviorAgent::basic_agent_pre_action()
{
      listen();
@
}



void NewBehaviorAgent::basic_agent_received_transmission(ReceivedTransmissionMessage *msg)
{
      if (m_performing_behavior)
```

```
        {
@
        }
}


void NewBehaviorAgent::basic_agent_answer_query(int observer_addr, QueryMessage *msg)
{
@
}


void NewBehaviorAgent::basic_agent_post_action()
{
        if (m_performing_behavior)
        {
@
        }
}


void NewBehaviorAgent::basic_agent_stop()
{
@
}
$
```

APPENDIX D: LOWPOWERNEWBEHAVIORAGENT CLASS FILES

```
/*=============================================================*/
/*                LOWPOWERNEWBEHAVIORAGENT.H                   */
/*  ---------------------------------------------------------  */
/*  This class uses multiple inheritence to instantiate a new  */
/*  low power behavior agent (LowPowerNewBehviorAgent) from    */
/*  the NewBehaviorAgent generated by the SwarmGenerator and   */
/*  the LowPowerAgent included in the swarm library.           */
/*  ---------------------------------------------------------  */
/*                                                             */
/*  Example:               BasicAgent                          */
/*                             |                               */
/*                  +---------+---------+                     */
/*                  |                   |                     */
/*              LowPowerAgent      NewBehaviorAgent            */
/*                  |                   |                     */
/*                  +---------+---------+                     */
/*                            |                               */
/*                LowPowerNewBehaviorAgent                     */
/*                                                             */
/*  ---------------------------------------------------------  */
/*  Author: Errol McEachron                 Date: 03/20/2002   */
/*  ---------------------------------------------------------  */
/*=============================================================*/

#ifndef LOW_POWER_NEWBEHAVIOR_AGENT_H
#define LOW_POWER_NEWBEHAVIOR_AGENT_H

#include "LowPowerAgent.h"
#include "NewBehaviorAgent.h"

class LowPowerNewBehaviorAgent : public LowPowerAgent, public NewBehaviorAgent
{
public:
      LowPowerNewBehaviorAgent(int my_addr, int world_addr, const Gate &world_gate,
                                              double sensor_sensitivity);
};
#endif
```

```
/*============================================================*/
/*               LOWPOWERNEWBEHAVIORAGENT.CPP                 */
/*  ---------------------------------------------------------  */
/*  This class uses multiple inheritence to instantiate a new  */
/*  low power behavior agent (LowPowerNewBehviorAgent) from    */
/*  the NewBehaviorAgent generated by the SwarmGenerator and   */
/*  the LowPowerAgent included in the swarm library.           */
/*  ---------------------------------------------------------  */
/*                                                             */
/*  Example:               BasicAgent                          */
/*                             |                               */
/*                  +---------+---------+                     */
/*                  |                   |                     */
/*              LowPowerAgent      NewBehaviorAgent            */
/*                  |                   |                     */
/*                  +---------+---------+                     */
/*                            |                               */
/*                LowPowerNewBehaviorAgent                     */
/*                                                             */
/*  ---------------------------------------------------------  */
/*  Author: Errol McEachron                 Date: 03/20/2002   */
/*  ---------------------------------------------------------  */
/*============================================================*/

#include "LowPowerNewBehaviorAgent.h"


LowPowerNewBehaviorAgent::LowPowerNewBehaviorAgent(int my_addr, int world_addr,
                                                                                const Gate &world_gate,    double
sensor_sensitivity)
       : BasicAgent(my_addr, world_addr, world_gate, sensor_sensitivity),
         LowPowerAgent(my_addr, world_addr, world_gate, sensor_sensitivity),
         NewBehaviorAgent(my_addr, world_addr, world_gate, sensor_sensitivity)
{ }
```

APPENDIX E: THE DISPERSE, CONVERGE, AND RESCUE CLASS FILES

```
#ifndef DISPERSE_AGENT_H
#define DISPERSE_AGENT_H

#include "BasicAgent.h"
#include "Position.h"

class DisperseAgent : public virtual BasicAgent
{
public:
	DisperseAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity);

protected:
	void basic_agent_start();
	void basic_agent_pre_action();
	void basic_agent_received_transmission(ReceivedTransmissionMessage *msg);
	void basic_agent_answer_query(int observer_addr, QueryMessage *msg);
	void basic_agent_post_action();
	void basic_agent_stop();

private:
	DisperseAgent(const DisperseAgent &disperse_agent);
	const unsigned int m_random_seed;

	bool m_dispersing;
	Position m_last_direction;
};


#endif
```

```
#include "DisperseAgent.h"
#include "Debug.h"


DisperseAgent::DisperseAgent(int my_addr, int world_addr, const Gate &world_gate,
                                                    double sensor_sensitivity)
       : BasicAgent(my_addr, world_addr, world_gate, sensor_sensitivity), m_random_seed(rand())
{

}


void DisperseAgent::basic_agent_start()
{
       srand(m_random_seed);
       m_dispersing = true;
}


void DisperseAgent::basic_agent_pre_action()
{
       listen();
}


void DisperseAgent::basic_agent_received_transmission(ReceivedTransmissionMessage *msg)
{

}


void DisperseAgent::basic_agent_answer_query(int observer_addr, QueryMessage *msg)
{

}
```

```
void DisperseAgent::basic_agent_post_action()
{
	if (m_dispersing)
	{
		//Disperse
		double x = ((((double) rand()) / RAND_MAX) - 0.5);
		double y = ((((double) rand()) / RAND_MAX) - 0.5);
		m_last_direction = Position(x, y);
		move(m_last_direction);
	}
}


void DisperseAgent::basic_agent_stop()
{

}
```

```
#ifndef CONVERGE_AGENT_H
#define CONVERGE_AGENT_H

#include "BasicAgent.h"
#include "Position.h"

class ConvergeAgent : public virtual BasicAgent
{
public:
	ConvergeAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity);

protected:
	void basic_agent_start();
	void basic_agent_pre_action();
	void basic_agent_received_transmission(ReceivedTransmissionMessage *msg);
	void basic_agent_answer_query(int observer_addr, QueryMessage *msg);
	void basic_agent_post_action();
	void basic_agent_stop();

private:
	ConvergeAgent(const ConvergeAgent &converge_agent);
	const unsigned int m_random_seed;

	bool m_converging;
	double m_power_current_msg;
	double m_power_of_last_received_msg;
	Position m_last_direction;
};


#endif
```

```
#include "ConvergeAgent.h"
#include "Debug.h"


ConvergeAgent::ConvergeAgent(int my_addr, int world_addr, const Gate &world_gate,
                                                    double sensor_sensitivity)
       : BasicAgent(my_addr, world_addr, world_gate, sensor_sensitivity), m_random_seed(rand())
{

}


void ConvergeAgent::basic_agent_start()
{
       srand(m_random_seed);
       m_converging = false;
}


void ConvergeAgent::basic_agent_pre_action()
{
       listen();
}


void ConvergeAgent::basic_agent_received_transmission(ReceivedTransmissionMessage *msg)
{
       m_power_of_last_received_msg = m_power_current_msg;
       m_power_current_msg = msg->get_power();

       if (m_power_current_msg > m_power_of_last_received_msg)
       {
              m_converging = true;
              fprintf(DEBUG, "message stronger\n");
       }
}


void ConvergeAgent::basic_agent_answer_query(int observer_addr, QueryMessage *msg)
{

}
```

```
void ConvergeAgent::basic_agent_post_action()
{
	if (m_converging)
	{
		//Converge
		m_converging = false;
		fprintf(DEBUG, "message adjust.\n");
	}
	else
	{
		//Adjust
		double x = ((((double) rand()) / RAND_MAX) - 0.5);
		double y = ((((double) rand()) / RAND_MAX) - 0.5);
		m_last_direction = Position(x/5, y/5);
		move(m_last_direction);
	}
}


void ConvergeAgent::basic_agent_stop()
{

}
```

```
#ifndef RESCUE_AGENT_H
#define RESCUE_AGENT_H

#include "BasicAgent.h"



class RescueAgent : public virtual BasicAgent
{

public:
	RescueAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity);

protected:
	void basic_agent_start();
	void basic_agent_pre_action();
	void basic_agent_received_transmission(ReceivedTransmissionMessage *msg);
	void basic_agent_answer_query(int observer_addr, QueryMessage *msg);
	void basic_agent_post_action();
	void basic_agent_stop();


private:
	RescueAgent(const RescueAgent &rescue_agent);

	const unsigned int m_random_seed;
	bool m_rescuing;
};



#endif
```

```
#include "RescueAgent.h"
#include "Debug.h"


RescueAgent::RescueAgent(int my_addr, int world_addr, const Gate &world_gate,
                                                    double sensor_sensitivity)
       : BasicAgent(my_addr, world_addr, world_gate, sensor_sensitivity), m_random_seed(rand())
{

}


void RescueAgent::basic_agent_start()
{
       srand(m_random_seed);
       m_rescuing = false;
}


void RescueAgent::basic_agent_pre_action()
{
       listen();
}


void RescueAgent::basic_agent_received_transmission(ReceivedTransmissionMessage *msg)
{
       if (msg->get_power()/200 > .99)
       {
              m_rescuing = true;
       }
}


void RescueAgent::basic_agent_answer_query(int observer_addr, QueryMessage *msg)
{

}
```

```
void RescueAgent::basic_agent_post_action()
{
	if (m_rescuing)
	{
		//Rescue -- Not sure what that means so just do nothing (stop).
		fprintf(DEBUG, "message RESCUE.\n");
	}
}


void RescueAgent::basic_agent_stop()
{

}
```

APPENDIX F: RELATIVE FILES FROM THE SWARM FRAMEWORK

```
#ifndef BASIC_AGENT_H
#define BASIC_AGENT_H

#include "Entity.h"
#include "ReceivedTransmissionMessage.h"
#include "QueryMessage.h"
#include "Position.h"


class BasicAgent : public Entity
{

public:
	BasicAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity);

	double get_sensor_sensitivity() const;

protected:
	// Behavior.
	void entity_start();
	void entity_pre_action();
	void handle_message(int sender_addr, Message *receive_msg);
	void entity_post_action();
	void entity_stop();

	void answer_query(int observer_addr, QueryMessage *msg);

	virtual void basic_agent_start() = 0;
	virtual void basic_agent_pre_action() = 0;
	virtual void basic_agent_received_transmission(ReceivedTransmissionMessage *msg) = 0;
	virtual void basic_agent_answer_query(int observer_addr, QueryMessage *msg) = 0;
	virtual void basic_agent_post_action() = 0;
	virtual void basic_agent_stop() = 0;

	// Stuff with non-functional properties.
	virtual void move(const Position &position) = 0;
	virtual void transmit(const char *data, double power, double frequency) = 0;
	virtual void listen() = 0;
```

```
        virtual bool listening() const = 0;

        double m_battery_level;


private:
        BasicAgent(const BasicAgent &basic_agent);

        double m_sensor_sensitivity;

};


#endif
```

```
#include "BasicAgent.h"
#include "BatteryLevelMessage.h"

#include "Debug.h"


BasicAgent::BasicAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity)
	: Entity(my_addr, world_addr, world_gate), m_sensor_sensitivity(sensor_sensitivity)
{
	add_observable_info_type(BATTERY_LEVEL_INFO);
}


double BasicAgent::get_sensor_sensitivity() const
{
	return m_sensor_sensitivity;
}


void BasicAgent::entity_start()
{
	m_battery_level = 100.0;

	basic_agent_start();
}


void BasicAgent::entity_pre_action()
{
	basic_agent_pre_action();
}
```

```
void BasicAgent::handle_message(int sender_addr, Message *receive_msg)
{
	switch (receive_msg->get_type())
	{
		case RECEIVED_TRANSMISSION_MSG:
			if (listening())
			{
				basic_agent_received_transmission((ReceivedTransmissionMessage *) receive_msg);
			}
			break;
		case QUERY_MSG:
			answer_query(sender_addr, (QueryMessage *) receive_msg);
			basic_agent_answer_query(sender_addr, (QueryMessage *) receive_msg);
			break;
		default:
			fprintf(DEBUG, "BasicAgent::handle_message - WARNING - don't understand message type
				%d.\n", receive_msg->get_type());
			break;
	}
}


void BasicAgent::answer_query(int observer_addr, QueryMessage *msg)
{
	if (msg->get_info_types() & BATTERY_LEVEL_INFO)
	{
		Message *battery_level_msg = new BatteryLevelMessage(m_battery_level);
		m_mailbox->send(m_mailbox->getHandle(observer_addr), &battery_level_msg, sizeof(battery_level_msg));
	}
}


void BasicAgent::entity_post_action()
{
	basic_agent_post_action();
}


void BasicAgent::entity_stop()
{
	basic_agent_stop();
}
```

```
#ifndef LOW_POWER_AGENT_H
#define LOW_POWER_AGENT_H

#include "BasicAgent.h"


class LowPowerAgent : public virtual BasicAgent
{

public:
	LowPowerAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity);

protected:
	void move(const Position &position);
	void transmit(const char *data, double power, double frequency);
	void listen();
	bool listening() const;

private:
	LowPowerAgent(const LowPowerAgent &low_power_agent);

	int m_listen_time;

	const double m_listen_cost;		// per unit time
	const double m_transmit_cost;	// per unit power
	const double m_move_cost;	// per unit distance

};


#endif
```

```
#include "LowPowerAgent.h"
#include "MoveEntityMessage.h"
#include "TransmitMessage.h"


LowPowerAgent::LowPowerAgent(int my_addr, int world_addr, const Gate &world_gate, double sensor_sensitivity)
       : BasicAgent(my_addr, world_addr, world_gate, sensor_sensitivity), m_listen_time(-1),
         m_listen_cost(0.001), m_transmit_cost(0.001), m_move_cost(1.0)
{ }


void LowPowerAgent::move(const Position &position)
{
       double cost = m_move_cost * position.distance();
       if (m_battery_level >= cost)
       {
              Message *move_msg = new MoveEntityMessage(position);
              m_mailbox->send(m_mailbox->getHandle(m_world_addr), &move_msg, sizeof(move_msg));
              m_battery_level -= cost;
       }
}


void LowPowerAgent::transmit(const char *data, double power, double frequency)
{
       double cost = m_transmit_cost * power;
       if (m_battery_level >= cost)
       {
              Message *transmit_msg = new TransmitMessage(data, power, frequency);
              m_mailbox->send(m_mailbox->getHandle(m_world_addr), &transmit_msg, sizeof(transmit_msg));
              m_battery_level -= cost;
       }
}
```

```
void LowPowerAgent::listen()
{
	double cost = m_listen_cost;
	if (m_battery_level >= cost)
	{
		m_listen_time = m_mailbox->getTime();
		m_battery_level -= cost;
	}
}


bool LowPowerAgent::listening() const
{
	return (m_listen_time == m_mailbox->getTime());
}
```