PREFACE

This project was especially interesting for me because it successfully combined my interests in computer programming and soccer. I have been doing computer programming for the last five years ranging from class projects to intercollegiate programming contests. In addition, I have played, coached, and followed soccer my whole life.

I would like to thank Professor David Evans for introducing me to swarm programming and the UVa RoboCup effort, and supervising this project. I would also like to thank Professor Bryan Pfaffenberger for what he has taught me to help me write this report. In addition, I would like to thank Yannick Loitiere for providing me with code to start the project with, providing an XML log file to work with, and answering several questions.

TABLE OF CONTENTS

TABLE OF DIAGRAMS

ABSTRACT

"Swarm" computing is a promising, state of the art area of research in computer science. It is a field with many possible applications, which makes it an important research topic. The basic definition is: programming a group of computing elements to work together and achieve some goal. With the recent rapid advancement in computer hardware, swarm computing has become a reality and now it is up to computer science to make use of these advancements.

The Department of Computer Science at the University of Virginia has been working with swarm programming by applying it to simulated soccer. The broad goal of this project is to aid in this research initiative. Swarm programming provides a special challenge in that most programs do not simply succeed or fail. There are many levels of success and failure which must somehow be measured. The specific objective of this project is to achieve this by developing a software tool that can evaluate the performance of a simulated soccer team's defense.

Given a XML server log created by a simulation, the tool evaluates the team's defensive performance. Five separate evaluation criteria were developed to do this. The software was developed with information hiding and modularity in mind, and is therefore easier to work with from a programmer's standpoint than less organized code. The software was also designed with flexibility and extensibility in mind so that future users can add or remove their own evaluation criteria as well as give weights to each one.

The resulting software has both advantages and disadvantages over manual evaluation. It is much faster, more flexible and extensible, and can provide quantitative analysis and results. However, it could require heavy computer usage, has a limited evaluation scope, and its correctness can only be determined through manual means. The software should be useful to programmers attempting to get an idea of how well their group behavior controlling software is performing. It provides a quantitative analysis of the behavior of a simulated group of objects so that the programmer may see the strengths and weaknesses of the algorithms in use. The short term goal is a tool to evaluate a specific type of swarm program, which the developed tool does. The long term desired result is better swarm programming and an increased potential of swarm computing. This cannot yet be determined, but the potential is there.

I. INTRODUCTION

Swarm programming is a currently evolving area of research in computer science. The possible benefits are numerous, and there is plenty of room for contribution. The goal of the project is to contribute a software product that will hopefully aid swarm programmers.

Problem Statement

"Swarm" computing refers to computing over a group of computing elements rather than one stand-alone computer. These computing elements look at the state of things around them and then decide what actions to take. Swarm programming is the actual programming of these elements so that they behave as desired to accomplish some group goal. However, research in the area is in its early stages. So much is being done in terms of developing swarm programs, but not enough is being done in terms of evaluating swarm programs. This evaluation is the focus of the project. By contributing this project, the objective is to aid developers and, ultimately, lead to better programs which can then be used to solve countless problems.

Rationale and Scope

A good way to go about developing better algorithms is by working with simulated groups of elements. Working with simulations rather than actual groups of

programmable objects eliminates many limitations and allows for easier testing and debugging. Creating simulations of group behavior is a difficult task by itself, however. Once a simulation is functional, there needs to be some way to evaluate it. Does it do what it is supposed to? In the case of soccer, it can be evaluated by the result of the game. Nevertheless, evaluating specific aspects of the team's performance is more difficult. Does it satisfy property A or property B? These are the types of questions that this project will help answer.

Currently, there is no standard method for evaluating swarm programs. The ideal evaluation method would be simple and quick to use and would provide quantitative data. These are two advantages of creating software to perform the evaluation. Software is customizable and can provide the simplest user experience possible as well as quantitative results. These results can then be analyzed in an attempt to improve the original swarm program.

The Department of Computer Science at the University of Virginia has been working with swarm programming by applying it to simulated soccer. The UVa RoboCup Team has their own simulated soccer team for which they have written the code. The broad goal of this project is to aid in this research initiative. Swarm programming provides a special challenge in that most programs do not simply succeed or fail. There are many levels of success and failure which must somehow be measured. The specific objective of this project is to achieve this measurement. Developing a software tool that can evaluate a simulation involving a group of programmed players is

the chosen method of doing so. Given a log created by a simulation, the tool is able to evaluate the group performance according to specific criteria. For the purposes of this project, the performance of a simulated soccer team's defense is evaluated.

The software is also designed so that it may be easily changed. This way, the user can add or remove evaluation criteria by adding their own functions or removing the current ones without causing any side effects. The software should be useful to programmers attempting to get an idea of how well their group behavior controlling software is performing. The tool provides a quantitative analysis of the behavior of a simulated group of objects (in this case the defense of the simulated soccer team) so that the programmer may see the strengths and weaknesses of the algorithms in use.

Organization

Chapter 2 provides background. Chapter 3 discusses the software design methodology used in the project. Chapter 4 gives the results. Chapter 5 interprets the results and provides recommendations.

II. SWARM COMPUTING

Computer programming originally consisted of applications that were fairly complicated, but lacked the capability to achieve more advanced tasks. As both the technological and theoretical aspects of computer science have advanced, so has the scope of computer programs. The once limited applications have been replaced by complicated projects that must accomplish many goals and satisfy many constraints. While computer programs have evolved over the last fifty years or so, one aspect that has not been taken into consideration is the changing computing medium.

Previously, programs ran on single stand-alone computers, whereas in 2000, less than two per cent of the computing units deployed worldwide were stand-alone computers [5]. Programs are starting to run on groups or swarms of computing elements, which may come in may forms besides the traditional computer. Advancements in areas like computer networking and computer architecture have made it possible for large tasks to be run by a group of computing elements each behaving independently, yet working together as a group. Further development of standards and methods for writing programs that can handle this type of environment is important so that this new technology can be fully taken advantage of.

Since swarm programming is a state of the art topic, the majority of the major discoveries and advancements have been recent accomplishments. Much of the underlying fundamental logic was developed several years ago, however. The concept of

intelligent behavior arising from environmental interaction goes at least as far back as the work of Herbert A. Simon in 1969 [8]. It was not until around 1985 that this concept was applied to autonomous robots. In 1985, Rodney Brooks [3] wrote an important paper about robot control and behavior in which he introduced two key ideas. The first of these was that robot activity can be thought of as behaviors as opposed to separate functional modules. The second idea was that a completely centralized control was not needed in the case of multiple robots. Each can behave independently, with the group still achieving the desired outcome if they are programmed correctly. By implementing these ideas, "systems take advantage of individual agents' situatedness to reduce or eliminate the need for centralized control or global knowledge. This reduces the need for complexity of individuals and leads to robust, scalable systems" [8: 1].

The idea behind swarm programming is that each element of a group is programmed with the same logic which is relied upon to decide what to do given the state of its surrounding environment. By providing each element of the group with all it needs to behave as desired, a manual control system is not needed. As applied to RoboCup soccer, "the cooperative behaviors result from the interaction of simple individual behaviors such as attraction to the ball, repulsion from obstacles, and patrolling of an area when the ball is not visible" [8:5] as opposed to a global control system.

The only area of debate appears to be which research strategy should be taken. Some computer scientists believe that the best way to develop group behavior programs is to work with groups. Still, as Maja Mataric [7:1] points out, others claim that this is

not feasible and that research “should first focus on the single-agent, single-robot case”. Since virtual or simulated objects are easier to work with than actual robots, that seems to be the most logical first step in developing algorithms for real robots or computing elements. Real hardware elements have issues such as inaccurate/incomplete sensor info, less than desirable communication quality, limited resources, and delayed feedback. These issues make them harder to work with than software, because software can eliminate or simulate any of these obstacles [7:1]. This project focuses on a simulated environment which will be much easier to work with.

The sources consulted agree that swarm computing is an area of computer science with great potential, and will surely revolutionize computing as it is known today. There are some promising areas of research that relate directly to group behavior and swarm programming. One such area is amorphous computing. Amorphous computing can be defined as “engineering prespecified, coherent behavior from the cooperation of vast numbers of unreliable parts interconnected in unknown, irregular, and time-varying ways” [1:2]. Put simply, amorphous computing gets sensible results out of computing elements working together in a seemingly insensible way. Amorphous computing takes the group behavior computing philosophy and applies it to less conventional groups of elements thereby expanding the possible applications of this technology.

A good example of an application currently being researched is cellular computing. The way this works is by constructing logic circuits within living cells through interactions between individual cells. Although cellular computing is currently

at the primitive stage of development, “progress here would open a new frontier of engineering which could dominate the information technology of the next century” [1:9].

Another promising area of research resulting directly from swarm computing is networked sensors. Networked sensors are sensors that coordinate amongst themselves to achieve a larger sensing task. This requires *sensor network coordination* which is achieved by local algorithms where “simple node behavior achieves desired global objective” [4:1]. Networked sensors can be used to gather information from previously unreachable destinations and require little maintenance, which makes them a desirable area of current and future research. While these two fields are by no means the only promising research paths, they provide two good examples of the possibilities that arise because of swarm computing.

Group behavior computing is a state of the art area of computer science. It is constantly evolving and has already led to several promising areas of current and future research. Now that the technology is available, swarm programming can really be utilized and applied to many fields including, but not limited to, those discussed earlier. Where this project fits in is evaluation of swarm programming algorithms in a simulated environment. This project will evaluate the behavior of a simulated group of elements according to some criteria, thus evaluating the algorithms that drive the group behavior. This software will help in improving swarm programs, which can then be applied to real life groups of computing elements.

III. METHODOLOGY

The software developed takes an XML server log as input and produces its own log of evaluation scores as output. This section discusses different aspects of this software.

Software Use

From the user's standpoint, the software is quite simple (see Fig. 2). As far as the user is concerned, the software only has the following steps:

1) The user runs the program, providing the XML file as a command line argument or allowing the default selection to be used.
2) They then wait as the program parses the XML, evaluates the data, calculates scores, and outputs them to a text log.
3) When the program completes, the user will be able to check the text log file created by the program which lists timestamp values and their corresponding scores.

Fig. 1 Software functionality from the user's perspective

Software Functionality

Meanwhile, the program will actually be following these steps (see Fig. 3):

1) Once the XML file is supplied and evaluation settings are determined, the program begins to parse the log.
2) The software extracts data and stores it. Data extraction and storage continues for the current timestamp until the next one is reached.
3) The program calls each of the five individual evaluation functions and records the scores.
4) It then comes up with a final overall score and prints it out to the text log file.
5) Steps 2) through 4) are repeated for each timestamp in the XML server log.

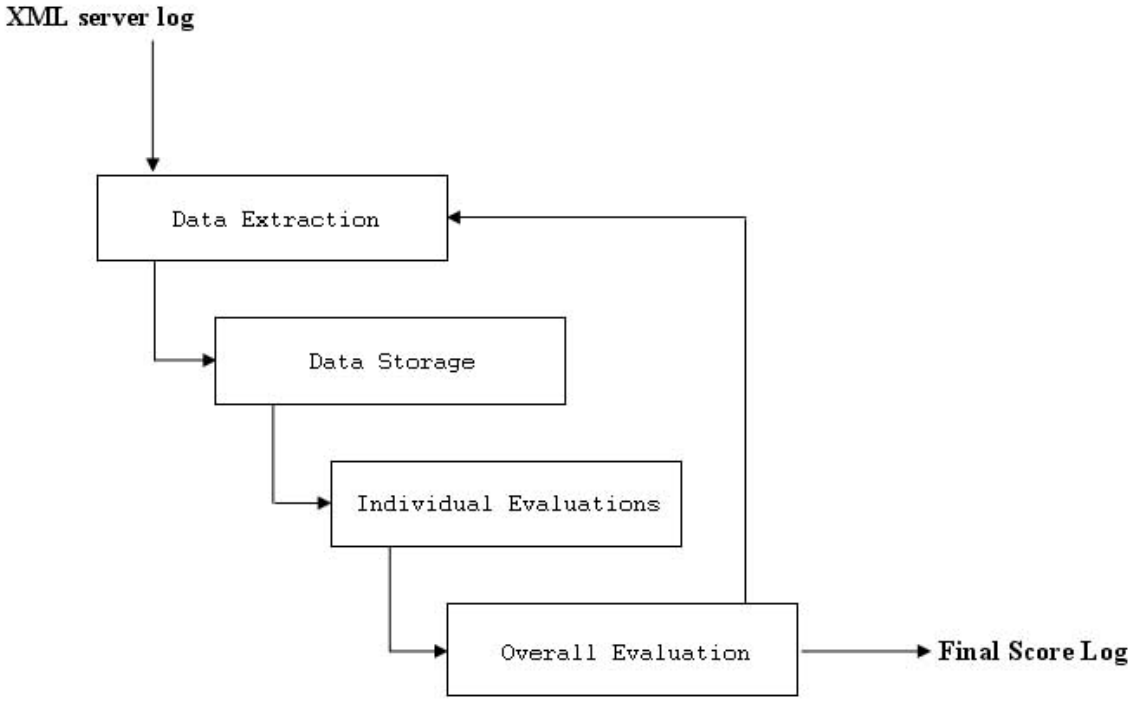

Fig. 2 Actual software functionality

Software Advancement

The software was designed with flexibility and extensibility in mind. The code is easy to understand and the user can add, remove, or change evaluation functions without causing the rest of the project to stop working. Several steps were taken to achieve this:

1) Comments were added at each potentially confusing or complicated piece of code so that anyone looking at the code can understand what it is doing.
2) Each evaluation function was made into a separate file. This way a function can be written completely separate from the rest of the project. Then all that is required to incorporate it into the project is to add the file and change a couple constant variable values
3) Common constants were all included in one file so that the user wouldn't have to search through the whole project just to change one thing.
4) The files were organized in a manner such that there are no circular includes and the data clearly flows from one to the next (see Fig. 2).

Evaluation Criteria

The evaluation criteria mentioned above are examined in more detail here. Only the simulated soccer team's defensive performance is evaluated. The reason behind limiting the scope is that the overall function of the software will not be altered, but the evaluation functions will be more detailed and therefore more useful. Being able to concentrate on one aspect of the team's performance as opposed to the team's overall

performance as a whole makes it possible to come up with more effective evaluation criteria.

Five evaluation criteria were considered:

1) **Balance1:** This evaluation functions tests whether the defense is balanced between the right, middle, and left side of the field. This criterion doesn't take the location of attackers into consideration, relying strictly on the location of the defenders. The more evenly balanced the defense is, the higher the score will be.

2) **Balance2**: This also tests whether the defense is balanced between the right, middle, and left side of the field. However, this criterion takes the location of attackers into consideration. Instead of testing whether the defense is evenly distributed across the field, it tests whether the defense is distributed in the same way as the offense.

3) **Numbers**: This criterion tests that the number of defenders is high enough for the number of attackers.

4) **Marking**: This function tests how well the attackers are marked by finding the distances between each attacker and their nearest defender.

5) **Pressure**: This tests how well the ball carrier is pressured by finding the distance between them and the nearest defender, the distance between them and the goal, and the position of the nearest defender in relation to them.

Each evaluation returns a score ranging from 0 to 100, where 100 is the best score possible. If an evaluation is irrelevant based on the current data, it returns a value of -1, letting the program know to ignore that particular instance of that evaluation. The first four evaluation criteria are executed using an imaginary line on the field. The line starts near the defensive goal and moves away from it, with the evaluation functions being called at each location of the line. The functions only take into consideration the players in between the imaginary line and the goal line. The score for each function is determined by reconciling the individual scores at each location of the line. This makes it possible to compare the scores at different distances from the goal. It also allows weights to be given, making the scores from closer to the goal count more than scores from further away from the goal.

IV. RESULTS

Once the implementation was complete, the software was tested with an actual XML server log. This section will discuss the outcome.

Process

A XML server log from a previously played game was obtained and fed to the program so that the results could be examined. The following statistics were recorded:

- The average evaluation scores for each of the five criteria for each team.
- The evaluation score for the UVA team for each criterion at every timestamp in which a goal is scored against them.

Averages and final scores were then calculated for the above data.

Results

The following average score data was recorded:

Average Scores

	UVA	FCP
Balance1	70.347	99.058
Balance2	65.753	55.599
Numbers	84.818	89.230
Marking	77.450	99.395
Pressure	73.119	96.731
Final	76.032	97.231

This data shows that the program evaluated the FCP team much higher than the UVA team. This makes sense since the final score of the game was 8 – 0 against the UVA team. The scores were rather uniform for the UVA team, but there was some

variation. The Balance2 score was lower than all the rest, showing that the UVA defenders were not balanced across the field in the same way that the FCP attackers were. In addition, the Numbers score was rather high, showing that UVA did have a good number of defenders to deal with the number of FCP attackers, but could not stop them from scoring.

The scores for the FCP team were almost all high, which is not surprising since their defense was never pressured by the UVA offense. The average Balance2 and Pressure for the FCP team can be somewhat misleading. These particular evaluations are only calculated if UVA attackers and the ball are in the FCP team's defensive half of the field. Since this did not occur often in the game, there is not really enough data to make a meaningful conclusion about the scores in these two categories. However, the other three categories do show that the FCP team's defense was solid by those criteria.

In addition, the following scores at times when goals were scored were recorded:

UVA Scores @ Times when goals are scored (8 total)

	Goal #1	Goal #2	Goal #3	Goal #4	Goal #5	Goal #6	Goal #7	Goal #8	Avg
Balance1	48.000	22.000	29.000	74.250	38.250	42.000	50.000	18.500	40.250
Balance2	75.401	37.464	80.663	68.428	83.748	81.501	76.775	86.445	73.803
Numbers	64.366	85.412	86.000	52.958	69.859	87.765	77.887	82.471	75.840
Marking	71.900	59.100	73.100	56.800	70.600	77.400	54.500	56.000	64.925
Pressure	33.333	N/A	N/A	70.000	53.333	66.667	50.000	N/A	54.667
Final	58.600	50.994	67.191	64.487	63.158	71.066	61.832	60.854	61.897

This data shows that the evaluation scores for the UVA team were lower than the average scores when goals were scored. This confirms what was expected. While four of the five evaluations returned lower than average scores here, Balance2 actually

returned a higher one. This suggests that the UVA team's defense did a good job of spreading the field in a way to match up with the opposing offensive players. However, since the scores in the other four areas were lower, that was not enough to prevent the other team from scoring.

Overall, the results were promising. Further discussion and interpretation will follow in the next chapter.

V. CONCLUSIONS

This final chapter will provide an interpretation of the results as well as recommendations for continued work on the project. In addition, it will discuss the success of the project as a whole and its significance.

Interpretation of Results

The specific objective of this project was to provide a tool to evaluate the defense of a simulated soccer team. The success of the project in this aspect can be determined by the actual results of testing the software. These results showed two major trends in the scores.

1) The scores were higher for the winning team.
2) The scores for the UVA team were lower on average at points when goals were scored.

Judging by these, the program can, to some extent, successfully analyze the performance of a team's defense. The second fact above is more important than the first because the scores for the winning team are less relevant than the scores for the losing team. Since the FCP team was not forced to play defense much in the game, the scores evaluating their defense are less meaningful. However, the fact that the UVA evaluation scores were lower when goals were scored is quite significant. Since the UVA defense was constantly being pressured, the average scores were more meaningful than they were for the FCP team as there was more data to work with. Since most goals occur, in part,

due to lack of defense, the fact that the scores were lower when goals were scored makes sense.

Since the scores the program calculates seem to make sense, the program can be used to analyze and improve the defensive performance of the UVA team. The swarm program that controls the team can be tweaked to achieve better scores from this tool. This, ideally, would, in turn, improve the "on-field" performance of the team.

The broader objective was to provide swarm programmers with a new method of evaluating the programs they write. That method is the use of software such as the tool developed for this project. After developing and working with such a tool, the benefits and drawbacks of this method can be analyzed.

Speed & ease of use. The software took approximately one minute to read in the server log file, evaluate the data, and output its own log file. Going through the XML server logs by hand would take so long it would be impractical. Watching the game play out is another way to evaluate the team's performance, but would also take significantly longer than using software. The numerous tests and calculations are only feasible if automated. By taking so little time to complete, software evaluation allows more time for further analysis to be performed.

Flexibility & extensibility. Using programmed evaluation criteria is flexible as well as efficient. Any criteria that can be transformed into code can be evaluated

using this software. This allows the software to test an infinite amount of properties. The ability to set weights for each function and to add or remove evaluation functions also makes the software flexible. The user can manipulate the evaluation criteria in several ways.

Quantitative analysis & results. This software provides quantitative analysis and results. Multiple evaluations can be made and compared to each other without any human error or bias. Quantifiable results provide a better foundation for analysis of the results. This analysis will, in turn, help improve the original swarm algorithms.

Evaluating swarm programs using software such as that developed for this project also comes with a few disadvantages.

Computer usage. In all, the program required about 16 megabytes of hard disk space. This included the input server log, the output logs, and the executable itself. The program also used up to 14 megabytes of memory at a given time. This data is from a log which had 2,834 timestamps. For a full soccer game, this equates to about one timestamp for every two seconds of play. If more timestamps were used, the hard disk and memory usage would increase, which could be an issue, depending on the computer that is running the program.

Limited evaluation scope. While the software can evaluate any properties programmed into it, that is all it can evaluate. Therefore the program will miss anything that wasn't explicitly programmed into it. This can include very simple properties that the programmer just did not think of, but would have picked up from watching the game play out.

Evaluation correctness. The correctness of the evaluations provided by this tool is dependent upon the programmer. If the programmer makes errors in coding, the evaluations will have errors too. The only way to truly test the evaluations is to watch the game play out and compare it with the evaluation scores to make sure they are consistent.

Recommendations

After reviewing the advantages and disadvantages of using this software tool to evaluate swarm programs, there are definitely some tradeoffs. An optimal solution seems to be a combination of the automated and manual approaches. The majority of evaluation criteria should be coded and inserted into software. However, the game, or replay thereof, should be analyzed by the user as well. This way the user does not miss simple evaluation criteria that were not programmed into the software. Also, the user can test the validity of the coded criteria by ensuring that it seems consistent with the game itself. While, at first, this seems to defeat the purpose of creating the software, in reality it far from does so. Watching games and replays a couple times for simple evaluations and to check the coded criteria takes far less time than the alternative. The alternative is

manually evaluating all properties by repeatedly watching the games, which would take significantly longer.

There are several advancements that would have been made to this software given more time, and that can still be made in the future. The goal was to create a functioning program which is simple to use, which would not have been possible if the following features were all included because they would have added too much development time.

User interface. One possible addition would be to add a user interface to the program. Currently, the program uses simple command line input and output. A nice interface would make the program a bit easier to use.

Evaluation criteria input. Another advancement that could be made is the ability to accept evaluation criteria as input. While this would take a significant amount of work, it would help users of the software. They would be able to add properties to be evaluated without having to know how to program, and without having to edit the software.

Generalized log input. The software could be changed so that it could read in a more generalized type of XML log. This way, the program could evaluate other simulations besides just soccer games. The tool could be very powerful if the mentioned additions were implemented.

Given the benefits and drawbacks of the software, it can be considered successful. The goal was to create specific software to aid swarm programmers by providing a means to evaluate their programs as well as explore the idea of software as a means of evaluation. This tool does that, and provides some distinct advantages over manual evaluation.

VI. BIBLIOGRAPHY

[**1**] Harold Abelson, Don Allen, Daniel Coorel, Chris Hanson, George Homsy, Thomas Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman and Ron Weiss. "Amorphous Computing". *Communications of the ACM* Volume 43, Issue 5, 2000.

[**2**] Guillaume Belson, Frederique Biennier, Beat Hirsbrunner. "Multi-Robot Path-Planning Based on implicit Cooperation in a Robotic Swarm." *Proceedings of the Second International Conference on Autonomous Agents*. Minneapolis, Minnesota. May 10 - 13, 1998.

[**3**] Brooks, Rodney A. "A Robust Layered Control System for a Mobile Robot." *MIT AI Lab Memo* 864, September, 1985.

[**4**] Deborah Estrin, Ramesh Govindan, John Heidemann and Satish Kumar. "Next Century Challenges: Scalable Coordination in Sensor Networks." *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks* (MobiCOM '99). Seattle, Washington, August, 1999.

[**5**] Evans, David. "Programming The Swarm." July 24, 2000.

[**6**] Dani Goldberg and Maja Mataric. "Robust Behavior-Based Control for Distributed Multi-Robot Collection Tasks," *USC Institute for Robotics and Intelligent Systems Technical Report IRIS-00-387*. July. 2000.

[**7**] Mataric, Maja. "Coordination and Learning in Multirobot Systems." *IEEE Intelligent Systems*, Mar/Apr 1998.

[**8**] Barry Werger and Maja Mataric. "From Insect to Internet: Situated Control for Networked Robot Teams", *Annals of Mathematics and Artificial Intelligence* 31:1-4, pp. 173-198, 2001.

[**9**] Werger, Barry. "Cooperation Without Deliberation: A Minimal Behavior-based Approach to Multi-robot Teams." *Artificial Intelligence* 110, pp. 293-320, 1999.

[**10**] Corten, Emiel. "SoccerServer Manual Ver.5. Rev. 00" http://www.dsv.su.se/~johank/RoboCup/manual/ver5.1release/browsable/main.html Last visited in March, 2002.

[**11**] Brogan, D.C. and Hodjkins, J.K. "Group Behaviors for Systems with Significant Dynamics" *The Journal of Autonomous Robots*. 1997.

[**12**] "RoboCup: What is Roboup" http://www.robocup.org/overview/21.html Last visited in March, 2002.

VII. APPENDIX

Appendix A: Code Listing