STARCAT AND ROBOCODE GENETIC ALGORITHM

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Lance W Finfrock

Fall 2008

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Lance W Finfrock:

Starcat and Robocode Genetic Algorithm

Joseph Lewis, Chair
Department of Computer Science

Jo Ann Lane
Department of Computer Science

Michael E. O'Sullivan
Department of Mathematics and Statistics

Approval Date

DEDICATION

I would like to dedicate this thesis to my Mother, Kathy, she is proof that it is never too late to do what you want, whatever that may be, and regardless of what others may think. I am so proud of her

"The vast majority of
intelligent behavior we see in nature comes not from
deductive reasoning or any such logical processes, but
from the ability to make good decisions in the face of
information that is simultaneously vast, incomplete, and
contradictory. "Good decisions" are those that give the
creature opportunity to continue behaving and hence make
other decisions." [1].

ABSTRACT OF THE THESIS

Starcat and Robocode Genetic Algorithm
by
Lance W Finfrock
Master of Science in Computer Science
San Diego State University, 2008

Starcat is a framework which provides the common functionality of Copycat, an autonomous analogy-making system. This functionality is capable of integration into many domains. This paper describes an experiment which integrated the Starcat framework into Robocode, a program that allows virtual user-modified robots to compete against each other in battles. The integration of these two frameworks resulted in an autonomous robot known as Botcat, which was used to prime a population of robots.

The Botcat robots were run through fitness tests in the Robocode environment, and then evolved using a Genetic Algorithm with roulette wheel selection. Many generations of Botcat robots were created using this technique in order to obtain optimal individuals which had autonomously learned how to excel in the Robocode environment. Emergent behavior was displayed from the robot populations, and some unexpected results were discovered. It was concluded that as the generations increased, the populations tended to perform better in the tests; however at the exceptionally higher generations the diversity in the population tapered and the populations' ability to evolve beneficial traits decreased. Various extensions of the study included comparisons involving fuzzy and crisp codelets, as well as alterations of the fitness tests to produce a more advanced Botcat individual. The results of the study are provided as well as suggestions for future experimentation with the Botcat system.

TABLE OF CONTENTS

PAGE

ABSTRACT vi
LIST OF TABLES ix
LIST OF FIGURES x
ACKNOWLEDGEMENTS xi
CHAPTER
1 INTRODUCTION 1
- Background 1
- Purpose of the Study 2
- Limitations of the Study 2

2 ARTIFICIAL INTELLIGENCE INTRODUCTION 4
3 BACKGROUND AND LITERATURE 7
- Starcat Overview 7
- Copycat Overview 9
- Madcat Overview 10
- Robocode Overview 11

4 METHODOLOGY 13
- Introduction 13
- Code Arrangement 14
  - Slipnet 14
  - Codelets 19
  - Fuzzy Codelets 19
  - Genetic Algorithm 23
  - Fitness Test 25
  - DNA 26

5 RESULTS 28
- Introduction 28
- Genetic Algorithm Improves Manually Created Robot 28

Unexpected Evolved Robot Traits..31
Fuzzy Codelets Increase Botcat's Perception..33
Moving Target..34
6 CONCLUSION..37
Genetic Algorithm Improves Manually-Created Robot 37
Unexpected Beneficial Traits...38
Fuzzy Codelets...40
Moving Target..40
7 FUTURE RECOMMENDATIONS..43
Introduction...43
Distance Detection Sensor...43
Evolvable Fuzzy Sets...44
Regressive Robot...46
Future Botcat Features...46
Short Term Memory: Dynamic Slipnet Nodes..47
Botcat Competing for Resources..47
Botcat Population Co-evolution...48
REFERENCES..50
A GLOSSARY..51
B ACRONYMS..56
C STARCAT QUICK REFERENCE..58
D HANDMADE BOTCAT SLIPNET..60

LIST OF TABLES

PAGE

TABLE 1. RESULTS OF FITNESS TESTS AMONG THREE GENETIC ALGORITHM EVOLVED BOTCAT POPULATIONS.. 30

TABLE 2. FITNESS TESTS RESULTS FROM BOTCAT POPULATIONS THAT WERE TESTED IN VARIOUS SIZED BATTLEFIELDS. POPULATIONS CONTAINED EITHER FUZZY OF CRISP CODELETS....................................... 34

TABLE 3. FITNESS TESTS RESULTS FROM MULTIPLE GENERATIONS POPULATIONS ATTEMPTING TO OBTAIN MOVING TARGETS OF VARIOUS SPEEDS... 35

LIST OF FIGURES

PAGE

FIGURE 1.REPRESENTS HOW FUZZY SETS ARE DETERMINED...................21

FIGURE 2.REPRESENTS HOW FUZZY SETS ARE DETERMINED...................45

ACKNOWLEDGEMENTS

Many people have provided me with inspiration, guidance, and support while I have worked towards my Bachelors and Masters in the field of Computer Science. Dr. Kendrick Mock at the University of Alaska Anchorage was the first Professor that helped me realize my passion in this field, and for that I am incredibly thankful. Dr. Joseph Lewis with San Diego State University sparked my interest in the field of Artificial Intelligence, and remains a great source of knowledge without which I would not have been able to achieve the research that I have performed in this study. My wife Rachel was a steady presence of encouragement throughout my education as well as my research, she has provided me with advice as well as a sounding board for my research ideas. There are many people that have made this both a life-changing and enjoyable experience, to all of you, I say Thank You

CHAPTER 1

INTRODUCTION

BACKGROUND

Many examples exist in the world involving the evolution and adaptation of a species due to environmental stimuli. These examples can be as obvious as a rat that learns to run a maze from beginning to end without encountering a negative environmental condition to as obscure as a gecko which has discovered that sitting near the light at night will allow it to obtain more moths. In both of these cases as well as the millions of others like them, an individual (or population) is required to adapt/evolve behavior as direct result of conditions that exist in their environment if they are to receive a desired resource (a piece of cheese or a moth as in the examples above). The behavior of an individual which arises directly due to the environmental stimulus and is not preprogrammed is known as emergent behavior (glossary in Appendix A). Emergent behavior is displayed in many species from insects to humans; it is necessary for a species to adapt. As the conditions within the environment change, species must alter their pre-existing behavior or develop new behaviors if they are to succeed.

The field of Artificial intelligence (AI), acronyms are found in appendix A, strives to produce systems that are capable of exhibiting signs of intelligence without the presence of human guided influences. This field has experienced a spectrum of both successful and unsuccessful attempts at this endeavor. The Physical Symbol Search Hypothesis by Newell and Simon was a major accomplishment which catapulted the field in a positive direction. From there, machines were produced which were capable of displaying behavior that would be predicted based upon the tools which the system was provided. Applications such as expert systems allowed a machine to resemble the ability to think, however with an understanding of the underlying system, it was clear that this behavior was reliant upon external input. Complex adaptive systems led the field of AI (acronyms in Appendix B) in a direction which would allow machines to display behavior which was unpredictable even to those that created the system.

An autonomous system was built by Douglas Hofstadter and Melanie Mitchell known as Copycat which is an analogy-making program that relied on the system building structures which would allow it to succeed in the domain [2]. Copycat utilizes a minimal set of initial structures that are provided to it in order to build unique structures that are relevant to the problem it currently needs to solve. The solution that Copycat provides is an example of emergent behavior; it is behavior that is displayed by the system and is not due to the presence of an outside source. Therefore even Hofstadter and Mitchell, the people who developed Copycat, would be unable to predict what the system's solution will be from one problem to the next.

The functionality of Copycat is something that could be applied to many systems, and was therefore developed into a framework for such purpose by Joseph Lewis. Starcat provides the means necessary for the Copycat functionality to be utilized across multiple domains [3]. This framework has been integrated into many applications; one notable is Madstar, a system which re-implements the Madcat system with Starcat as the underlying framework. The Madcat system uses the Copycat like framework to allow a robot the ability to develop maps of its surrounding environment in order to navigate successfully [4].

PURPOSE OF THE STUDY

Robocode is an application that provides a basis for user-created robots to compete against other robots in a battlefield environment. The framework of Robocode provides the ability to develop a robot and control parameters such as the size of a battlefield, the presence of objects within the battlefield, and the rules of a battle, such as time constraints. The purpose of this study was to integrate the Robocode and Starcat Frameworks into one system which, along with a Genetic Algorithm, would evolve populations of autonomous robots through many generations. The goal of the study was to evolve robots which were capable of using the basic Starcat functionality to develop the basic skills required to conduct themselves in the Robocode environment.

LIMITATIONS OF THE STUDY

It was unknown at the beginning of this study if many necessary factors for success were even a possibility. Creating a framework from the existing Robocode system was a top priority. If the system was not capable of integrating with Starcat, the study would come to a complete standstill. Upon successful integration of the two frameworks it was necessary to test hand-created robots to determine what pre-existing functionality is required for the system to head in a direction capable of producing autonomous behavior. This requirement would be the most important aspect of initializing the experiment. It is essential to produce an initial robot that possesses enough diversity that it is capable of priming a population with the means to autonomously behave; at the same time it is important to not cross the line of providing the robot with too much pre-existing functionality which would prevent future behavior from being considered emergent.

CHAPTER 2

ARTIFICIAL INTELLIGENCE INTRODUCTION

The field of Artificial Intelligence (AI) gained modern realization with the creation of the Physical Symbol System Hypothesis (PSSH) by Allen Newell and Herbert Simon. The PSSH states that “A physical symbol system has the necessary and sufficient means of general intelligent action” [5]. A physical symbol system is defined as being a system that manipulates individual symbols into varying patterns so that meaning can be drawn from the combined formation of these symbols. Each symbol has its own unique meaning, however the system will rearrange symbols so that intelligence is said to be exhibited from the combination. A computer is a physical symbol system therefore, according to the PSSH, it has the means necessary to exhibit general intelligence. The PSSH was first implemented as a search. To understand a search, consider a system faced with a problem which has many possibilities for being solved. If this problem were placed in a tree, the initial problem the system seeks to resolve would be at the root of the tree. Each child branch of that root represents a choice that the system could potentially make in order to find an answer to the search that is being performed. Each of those children will themselves have children and so on. When a search begins the system is presented with a problem. Every possible action that can be performed to find a solution to that problem is considered, from there the system exploits all possibilities that could occur based upon performing those first initial actions and so on. Although there are many possible paths that the system can traverse, few of those will be the most advantageous solution. Upon exploration of all possible outcomes to a predetermined depth in the search tree, the action with the most optimal outcome would be selected. This search technique was found to be useful for problems which possessed a discrete number of possible actions. Board games such as Chess and Checkers present examples of problems that could be solved by utilizing a search to explore each of the distinct actions that a player can perform. The search tree would be filled with options that a given player could make based upon the current board configuration. Navigating deeper down the search tree will generate the results of performing the actions on the branches of the path leading to the current position in the tree. The greater depths in the search tree represent further points in the game. When this search technique is employed for problems that possess an unbounded number of actions it becomes useless; the search cannot be completed in a reasonable amount of time.

Following the development of the PPSH was the Expert System, a system that is capable of providing information on specific topics. Each Expert System is designed specifically for one domain. The system utilizes information that is obtained from many experts in the field of the system's domain subject matter. For example, there may exist an Expert System which contains knowledge provided to it by the world's leading heart surgeons. This system would have the ability to perform searches on heart surgery based and provide answers to related queries. This system is useful in that is makes information that would only be known by few people now available to many. One of the many problems with Expert Systems was that they are limited to specific domains; if a system were asked a question that lay on the fringes of its domain, the system would not work correctly.

The history of AI is a progression of both successful and unsuccessful advances at revealing the connection between that which provides humans the capacity to think and implementing machines which possess this same ability. Many argue that a machine's ability to utilize a programmed set of rules is not an example of cognition, but merely the result of following a set of procedures. The study of Complex Adaptive Systems (CAS) presents the idea of Emergent Behavior being produced from machines. Emergent behavior is behavior that is displayed or produced by a system that is not predefined or programmed into the system from an outside source. In a sense, this behavior has come about without human involvement, and it can not be predicted or planned for. Emergent Behavior develops from past interactions that a machine has with its environment; these experiences allow it to develop new representation which it can utilize in future endeavors. "…representation occurs as a product of the active construction of experience from perception." [4] Representation is the concept of how knowledge is stored or represented by a system.

To give an example of how representation is developed in humans let us consider a child developing the concept of numbers. If you were to draw the number three on a board in front of children that have no concept of what the number three is, the drawing would be nothing more than a squiggly line to the kids. Now if you were to lay three cookies out on the table in front of the kids, due to past experience they would know that these are cookies. By utilizing objects that the children have already developed representation for this can assist in the process of developing new representation. By telling the children that the squiggly drawing on the board represents the number of cookies that are on the table, the children would be able to develop the concept of the number three. In the future the children will no longer view the number three as a squiggly line, but as a number that has meaning. This same process of developing concepts based on environmental interactions and past experiences is how, in the study of CAS, systems are capable of displaying Emergent Behavior. “A shift of focus in recent years, generated in part from the study of complex adaptive systems, has motivated research into the dynamic processes underlying [representation building structures].” [6]

The concept of CAS is a new enterprise of AI, and is represented in such systems as Starcat, Copycat, and Madcat (respectively described below). The study defined in this thesis paper utilizes CAS concepts in order to generate measurable emergent behavior whose evolution can be traced through multiple experimental generations.

CHAPTER 3

BACKGROUND AND LITERATURE

STARCAT OVERVIEW

Starcat is a framework which was created by Dr. Joseph Lewis to provide the basic functionality of the Copycat program; it allows other domains to utilize the essential functionality of Copycat. "Emergent representation is a key feature of [Starcat's] functioning and adaptive behavior is the desired result that it supports." [3] Unlike Copycat, the Starcat framework is not specific to any domain; it allows custom domain-specific modules to be added on top of the main Copycat engines. Lewis states that the Madcat and Copycat systems have "two important limitations… their inability to adapt to novel situations and their lack of true autonomy." [3] Aside from not having domain-specificity, Starcat also adds value to existing CAS in that it "is an open-ended architecture for computational systems that autonomously adapt their behavior to continuously changing environments." [3] Starcat's ability to allow for autonomous systems presents a new niche in the field of AI.

Machines that were limited both by their domain and their ability to learn and adapt are now capable of depicting emergent behavior in the face of an ever-changing environment. This revelation allows systems to develop representation much the same way that a child would interact with its environment, make decisions, and learn from its experiences. This novel ability provides the potential platform for the AI field to expand in a direction that lacks human guidance and control. This may be the basis for machines to exhibit intelligence without the underlying regulation of programmed control.

The main structures of the Starcat Framework are the Workspace, Slipnet, and Coderack (quick reference in Appendix C). The Workspace is the environment where objects are observed and analyzed. The Slipnet is a collection of connected nodes which represent concepts pertaining to the environment. The Coderack is a collection of commands from the Slipnet which are to be performed on objects in the Workspace. All three of these structures are continuously working together in order for the Framework to allow the domain in which it is applied obtain a full understanding of its environment.

The Slipnet is a network of nodes that transmit activation back and forth between each other via links. Each node in the Slipnet represents a concept that is specific to the environment of each unique domain that the Starcat Framework is utilized for. Examples of some concepts which may occur in an environment which requires movement are “backwards” or "forward”. An environment that requires testing to occur may contain nodes whose concepts are “pass” or “fail”. Nodes in the Slipnet contain a value called activation; this value ranges from 0 to 100. As a node’s concept is increasingly realized by the system its activation value will be increased as well. Realization of a concept means that in the Workspace that the concept occurs or is most likely occurring. If the environment is tested to see if a wall is present in the forward or backward position, and a wall is determined to be in the forward position, then the node for the concept of “forward” is realized and that node will receive a large portion of the activation points (let this value be X) compared to the node for the concept of “backwards”. The “backward” node may still receive a small portion of the activation points(100-X); this is explained in the Fuzzy Codelet section of Chapter 4. Slipnet nodes are interconnected and contain links which transmit activation from one node to another. This transmission of activation represents the realization of a node’s concept from the realization of another node’s concept. When the activation value of a Slipnet node reaches a high enough level (a pre-determined value), it will produce codelets that will be placed in the Coderack.

The Coderack is a stochastic queue of codelets; it is used to add the element of randomness to the placement of the Codelets into the environment. Codelets are placed onto the Coderack using a priority value; higher priority codelets have a better chance of being chosen over the other codelets in the Coderack. A Codelet’s priority value is determined by the amount of activation that it receives from its corresponding Slipnet node. When a Codelet is chosen this means that it will be placed into the Workspace.

A Codelet is a command or observer that is placed into the Workspace in order to interact with the environment. Codelets could be placed into the environment in order to detect such things as walls, obstacles, and other applications within the system. The information that is returned to the system from the Codelets’ observations activates a chain of events which involve both the Slipnet and the Workspace.

The Workspace is the environment where Codelets observe and perform action on the environment. If the domain that Starcat were utilized for were a car racetrack, the Workspace would be the actual racetrack. Codelets could exist in the racetrack Workspace in order to observe and report back on such actions as speed, the presence/absence of vehicles, or the location of a given vehicle on the racetrack. Each Workspace is unique in each domain in which Starcat is run, therefore modifications must be made to the Slipnet and Coderack according to the environment. It would be unnecessary for a node to contain the concept of "left" in a Workspace that dealt with musical sounds.

COPYCAT OVERVIEW

Copycat, an analogy-making program, was created by Douglas Hofstadter and Melanie Mitchell. Analogy-making can be defined as "the perception of two or more non-identical objects or situations as being the 'same' at some abstract level." [2] Copycat's domain is specific to the micro-world of letters of the English alphabet and their relationship to each other. The input into the Copycat program is three sets of letters, each set consisting of three letters. The first two sets of letters represent a transition that has occurred between the first and the second set. (The rationale behind the third set is explained in the following paragraph.) An example of input provided to Copycat for the first two sets could be ABC and ABD. These two sets symbolize the transition from the letters ABC to ABD. The rule of this transition could be: take the last letter of the first set, "C", and swap it with the next consecutive letter of the alphabet, "D", thereby resulting in the second set. The rule could also be to always replace the last letter in the first set with the letter D. Copycat was designed to choose the rule that fits best, which would be the first rule in the previous example, to swap the last letter in the first set with the next consecutive letter of the alphabet.

Copycat uses unique components that interact with each other in order to determine what action has taken place to transition between the first input set of letters and the second set. The components utilized to determine the transitioning action are part of the Starcat Framework and are described in detail in Chapter 3: Code Arrangement. Copycat's micro-world of letters of the alphabet domain allows such transitions as reversing the letters in the set, swapping letters within a set, changing whether a letter is displayed as capital or lowercase, duplicating letters, and so on. The action that Copycat determines to have taken place between the two sets is known as the transition rule.

To demonstrate its understanding of the relationship between the first two sets of letters, Copycat performs the transition rule on the third set of input letters. The result of this operation is the output. To continue with the example above (set 1: ABC, set 2: ABD), the third input set is EFG; if Copycat were to choose the most reasonable transition it would have a high chance of outputting EFH. This output would demonstrate Copycat's ability to model the transition between the first two input sets by recognizing the rule that the last letter in the set should be replaced with its successor in the alphabet. Another possible response from Copycat could be EFD, where Copycat determined the transition rule is to replace the last letter in the set with the letter D.

The main structures involved in Copycat are the standard features of the Starcat Framework: the Workspace, Slipnet, and Coderack. These structures are applied to each application described in this chapter.

MADCAT OVERVIEW

Dr. Joseph Lewis and colleagues at the University of New Mexico performed an experiment using a Copycat like framework to create an autonomous robot. "Robots with greater flexibility, adaptability, and robustness are possible using such an architecture. These would be useful in situations where a robot must autonomously adapt to its changing surroundings, for instance in unmanned space landings." [6] This experiment contributes to building a foundation of AI applications that apply to real-world situations or needs. The architecture is versatile in that it can be functional in both virtual as well as physical applications. The Copycat like framework later became the basis for the Starcat framework.

Madcat is capable of moving by means of wheels, and it interacts with and studies the environment through both "bump" and "sonar" sensors that take "snapshots" of the evironment. [6] The components of the architecture utilize readings from these sensors in order to determine the features present in Madcat's surroundings. A unique feature of the Madcat system is mapnet; this is where environmental features that are consistently observed in the snapshots are stored. Mapnet works to build a map of Madcat's environment. [4]

The ability of the Madcat system to interact with actual real-world environments allows it to build representation that can only come from the spontaneity and diversity that exists in the real world. In comparison, a computer-based system is only capable of experiencing a finite number of opportunities to build representation from; the greater the level of diversity in the environment in which the system interacts, the more robust a system will become.

ROBOCODE OVERVIEW

Robocode is an open source framework which was created by Mathew Nelson at IBM. The Robocode project is intended to provide an entertaining environment for individuals to learn the Java language. Code is written for Robocode in order to control robots in an environment where they are able to interact with each other.

Robocode provides a famework which allows a developer to create a customized robot. The framework is provided as a code library jar file. The basic class in this library is the Robot class, which, when extended, creates a new robot. The Robot class contains many different methods which can help to customize the behavior of a robot. There is one method in particular that needs to be created by the developer; this method is called “run”. The “run” method should be placed into a loop that is constantly observing and reacting to the environment. This loop ends when the robot loses all of its energy. Each robot begins with an initial 100 points of energy. Energy is lost when a robot runs into another robot or wall, gets shot by a bullet, or shoots a bullet. Energy is gained when a robot successfully shoots another robot.

Each robot has three separately moving parts: body, turret, and radar. The body of the robot moves similar to a tank on tracks, and is capable of turning left or right and moving forward or backward. An arc movement can be created by simultaneously turning a direction and moving either forward or backward. The turret piece on the robot can freely turn left or right 360 degrees. The radar component is used to view the locations of other robots and bullets in the robot’s environment. The radar component can freely turn left or right 360 degrees. Below is a list of some of the methods that the Robot class provides in order to allow a robot to react to its environment:

- ahead(double distance) - Moves the robot ahead (forward) by a distance measured in pixels.

- back(double distance) - Moves the robot backward by a distance measured in pixels.
- fire(double power) - Fires a bullet in the direction that the turret is facing.
- getX() - Returns the X position of the robot on the battlefield.
- getY() - Returns the Y position of the robot on the battlefield.
- turnLeft(double degrees) - Turns the robot's body to the left by the number of degrees passed in.
- turnRight(double degrees) - Turns the robot's body to the right by the number of degrees passed in.

The environment where robots are placed to interact with each other is in the Robocode battlefield. This battlefield is a rectangle with a width and height that can each be set to anywhere from 400 to 5,000 pixels. The dimension of the battlefield is predefined by the user prior to the beginning of a battle, and remains static throughout the battle. Each robot can move anywhere on the battlefield, with the exception that two robots cannot be in the same place simultaneously. A battle consists of two or more robots placed onto a battlefield. A battle ends when there is only one robot left in play.

CHAPTER 4

METHODOLOGY

INTRODUCTION

The motivations for using Robocode combined with the Starcat Framework for this study were many. The Robocode framework possesses many features which would allow Starcat to seamlessly integrate in order to create a truly unique autonomous environment similar to those of Madcat and Copycat. Starcat provided the potential to craft the functionality of Robocode into being a computer run application as opposed to requiring outside influences in order to operate. This chapter is dedicated to describing how the Starcat components as described in Chapter 3 were modified to Robocode's unique environment in order to create Botcat, the robot created for this experiment.

The components in this project were created in Java in the Eclipse IDE. The Java language was the optimal choice given that the Robocode and Starcat frameworks are both written in this language. The Eclipse IDE was chosen due to prior familiarity and proven compatibility with both the Starcat and the Robocode frameworks.

The Robocode framework was not modified, no code was changed in its codebase; this facilitated using the most up-to-date versions of the code as it was released. Upon installation, Robocode provides a specific folder in which to place Java classes that are used to control robots within the Robocode framework. The Starcat framework and the custom code created for this project were placed into this folder. When running the Robocode program, the Starcat robot, Botcat, can be selected as one of the robots to be placed in the battlefield environment.

There were many elements of the Botcat system that were either designed specifically for this study or that were greatly modified to Botcat's distinctive domain. The following sections in this chapter will describe the methods that were used to modify Starcat's framework and the integration of this into the Robocode framework. Descriptions of components that were utilized in this study are further explained, including Fuzzy Codelets and the User interface arrangement.

After customizing each component of both frameworks and successfully creating a fully functioning Botcat system, adjustments were made to certain components, and experiments were run on the populations of Botcats that were created. These experiments were executed to evolve the Botcat system and find the most optimal Botcat for success in the Robocode environment. Genetic Algorithms that utilized DNA specific to Botcat were utilized along with fitness tests in order to evolve/discover the most optimal Botcat system. This chapter explains creation and use of the Genetic Algorithm, DNA, and fitness tests used in this study.

CODE ARRANGEMENT

Upon starting up BotCat, Starcat's Slipnet, CodeRack, and Workspace components are initialized. Starcat is merely a framework that is comprised of these three components. The emergent behavior that is displayed by BotCat is the result of custom elements which were designed specifically for each Starcat component. With every environment that the Starcat framework is utilized for it is necessary to tailor the program. The following sections explain the modifications that took place to each Starcat component in order for the BotCat robot to successfully interact with its environment.

Slipnet

The Slipnet contains thirty-three custom Slipnet nodes; these nodes were created in a triple layered approach with each layer depending on and interacting with the functionality available in each of the other layers (display in Appendix D). The first layer contains nodes which are primitive, they represent simple ideas and have direct connections to the environment or battlefield. The second layer contains basic concepts which are derived from the first layer and represent slightly more complex ideas. The third layer combines the basic concepts from the second layer in order to make decisions as to which actions should be performed in the battlefield. These layers only exist to assist with understanding the system.

The following are the nodes which are present in the first layer along with the concepts they represent in terms of Botcat's position within the battlefield:

- obstacleLeft – an obstacle to the left.
- obstacleRight – an obstacle to the right.
- obstacleFront – an obstacle in front.

- obstacleBackward – an obstacle behind.
- clearleft – no obstacles to the left.
- clearRight – no obstacles to the right.
- clearFront – no obstacles in the front.
- clearBackward – no obstacles behind.
- targetRight – the target position is to the right.
- targetLeft – the target position is to the left.
- targetForward – the target position is in the front.
- targetBackward – the target position is behind.

The above Slipnet nodes receive activation directly from sensors that test the environment; these sensors are Codelets that are contently added by observer Slipnet nodes that test the Workspace. The first eight Slipnet nodes listed above create the basis of the Obstacle Avoidance System (OAS), while the other four nodes provide a basis for the Target Tracking System (TTS). When combined, these two systems provide the robot with the information necessary to make decisions about where to move within the battlefield. The basis for determining the amount of activation that is received by each node is explained further on in the Fuzzy Codelet section of this paper.

The OAS's basis collects the inputs of where obstacles in the battlefield are located relative to the robot. In addition to the eight nodes that comprise the OAS, there are four Observer Slipnet nodes (obstacelLeftObserver, obstacleRightObserver, obstacleFrontObserver, and obstacleBackwardObserver) that are continuously producing Codelets in order to detect obstacles and send feedback to the basis nodes. Each of the Observer Slipnet nodes' codelets are tied to two of the OAS's basis Slipnet nodes. Upon testing the environment, Codelets send feedback to their respective Slipnet nodes. Feedback is provided in the form of activation which is added to the appropriate nodes. As in Copycat, activation is a value that ranges between 0 and 100 points. If the concept that a node represents has been recognized within the environment, that node is more likely to receive a higher activation value.

The following is an example of an Observer feedback relationship. The Observer Codelet (obstacleFrontObserver) is tied to two nodes: obstacleFront and clearFront. The obstacleFrontObserver Codelet tests Botcat's forward direction of the environment for obstacles. If the Observer sees an obstacle in front of it, the majority of the available 100 activation points will be added to the obstacleFront node while a small percentage of the points will be added to the activation value of the clearFront node. The opposite happens when the observer does not detect an obstacle in the forward position. The basis of this system allows Botcat to remain informed of obstacles within its vicinity.

The TTS's basis collects inputs as to where the target is positioned relative to the robot. The TTS is comprised of targetRight, targetLeft, targetForward, and targetBackward Slipnet nodes. Similar to the OAS's basis, there are two observer Codelets (targetRightLeftObserver and targetForwardBackwardObserver) which detect the direction of the target and send feedback to their respective Slipnet nodes. Again, there are two Slipnet nodes that each observer Codelet connects to from the system. As predicted based on the names - the targetRightLeftObserver is connected to the targetRight and targetLeft nodes, and the targetForwardBackwardObserver is connected to the targetForward and targetBackward nodes.

The following example demonstrates how the Codelets and nodes are interconnected, and how information is traversed within the TTS. The targetRightLeftObserver tests the current direction of the target. If the direction of the target is found to be immediately to the left of the robot the targetLeft Slipnet node would receive 100 points of activation and the targetRight Slipnet node would receive 0. If the target's direction is northeast of the robot at an angle of 45 degrees, then the targetLeft Slipnet node would receive 25 points of activation and the targetRight Slipnet node would receive 75 points of activation. The targetForward and the targetBackward Slipnet nodes work in the same way. Utilizing all four of the Slipnet nodes, the direction of the target is determined and the activation is passed directly to the third layer in the Slipnet.

The second layer of Slipnet nodes contains basic concepts which were derived from the first layer. The second layer only pertains to the OAS; it creates concepts based upon what conditions are present in the battlefield. Below is a list of the nodes within the second layer along with a description of where they represent Botcat being positioned within the battlefield.

- backRightCorner – in a corner with obstacles to the right and back of it.
- backLeftCorner – in a corner with obstacles to the left and back of it.

- frontRightCorner – in a corner with obstacles to the right and front of it.
- frontLeftCorner – in a corner with obstacles to the left and front of it.
- backWall – obstacles only to its back.
- rightWall – obstacles only to its right.
- leftWall – obstacles only to its left.
- frontWall – obstacles only to its front.
- openSpace – no obstacles to any side of it.
- backFrontTunnel – in a tunnel with obstacles to the front and back of it.
- leftRightTunnel – in a tunnel with obstacles to the left and right of it.
- leftRightFrontCove – in a cul-de-sac with obstacles on the left, right, and front of it.
- leftRightBackCove – in a cul-de-sac with obstacles on the left, right, and back of it.
- leftBackFrontCove – in a cul-de-sac with obstacles on the left, back, and front of it.
- rightBackFrontCove – in a cul-de-sac with obstacles on the right, back, front of it.
- closeIn – obstacles on all sides of it.

The above second layer Slipnet nodes receive their activation from the first layer of Slipnet nodes by means of links. In these links, certain nodes of the first layer are connected to certain nodes of the second layer. These links are directional; the activation travels from the first layer to the second. The following is an example of node connections from the first layer to the second layer: the obstacleRight, obstacleLeft, clearBack, and clearForward nodes from the first layer are linked to the leftRightTunnel node from the second layer. The links between the first layer nodes and the second layer node would each have a length of twenty-five (the shorter the length the more activation that is sent), thereby leading each node on the first layer to transmit seventy-five percent of their activation to the node on the second layer.

Formula for determining the activation that is added to the receiving node:

$(100 – X) * Y = Z$

- X is the length of the connecting link
- Y is the activation of the sending node
- Z is the amount of activation added to the receiving node.

The higher the activation level of the node on the second layer, the more stable the concept of that node becomes. Each node represents a concept; for example the leftRightTunnel node represents the concept of an obstacle being to Botcat's left and right with no obstacles in the forward and back directions. When the leftRightTunnel's activation becomes high, Botcat will conclude that it is in a tunnel with walls to the right and left of it. Once Botcat establishes the configuration of the surrounding environment, it will proceed accordingly. The third layer of nodes comes into play once Botcat performs the actions that are deemed necessary given the environmental conditions present.

The third layer of the Slipnet is comprised of actions which are based upon concepts from the first and second layers. The third layer is a combination of both the OAS and the TTS; it is the performance layer. The nodes within this layer produce Codelets which are created in order to command Botcat to perform specific actions. Below is a list of the nodes in the third layer along with a description of the action they cause Botcat to perform.

- turnRight – turn right.
- turnLeft – turn left.
- moveForward – move forward.
- moveBackward – move backward.
- doNotMove – not move.

Activation of the above nodes allows the BotCat system to carry out its primary purpose of avoiding obstacles (OAS) and retrieving targets (TTS). Combined, the OAS and TTS enable emergent behavior to be exhibited by Botcat. BotCat is not preprogrammed with any skills that are necessary to function properly in a Robocode battle; it is imperative that emergent behavior is exhibited by the system in order for BotCat to succeed. Links exist from many concepts in the second layer to the third layer within the OAS in order for BotCat to perform certain actions. These links allow Botcat to distance itself from obstacles. For example, if upon testing the environment BotCat concludes that it is in a tunnel with obstacles on its right and left, it tends to move forward in order to get out of the tunnel.

The TTS creates a tendency for the robot to move closer to targets. This system is comprised of nodes solely within the first layer; concepts from the second layer are not needed. Nodes from the tracking system on the first Slipnet layer are linked to the third layer nodes. These connections are more intuitive than those in the OAS, they are the following: (targetLeft, turnLeft), (targetRight, turnRight), (targetForward, moveForward), and (targetBackward, moveBackward).

When combined, it is possible for there to be conflicts between the TTS and OAS in terms of what they want accomplished. For example, if there is an obstacle between Botcat and a target, the TTS may attempt to move forward (given that its purpose is to obtain targets) while the OAS will be attempting to move backward (it works to avoid obstacles including targets). These combined efforts produce the emergent behavior of Botcat.

The following example summarizes the interactions of the three Slipnet layers. This is an illustration of the chain of events that occurs prior to Botcat turning right in the battlefield. Assume that Botcat is in a position on the battlefield where there is a wall to its left and all of the other directions are clear. This situation would activate the obstacleLeft, clearRight, clearFront, and clearBack Slipnet nodes of the first layer. The leftWall Slipnet node (second layer) would receive activation from the obstacleLeft, clearRight, clearFront, and clearBack nodes (first layer) because it is connected to all four of these. Activation would then spread from the leftWall Slipnet node (second layer) to the turnRight Slipnet node (third layer). Once the turnRight Slipnet node receives enough activation points to become activated, it will add a turnRight Codelet to the Coderack. When this Codelet is selected to perform it sends a message to Botcat to “turn right”. The above process happens concurrently with observations and activations from the other nodes taking place.

Codelets

Observations of the battlefield environment are made by Codelets; Codelets are objects that interact with or observe the environment. This experiment utilized Codelets only for the purpose of observing the environment. A total of six Codelets were used, four for detecting obstacles (OAS) and two for detecting the direction of the target (TTS). The interaction that these six codelets had with the system will be explained later, it is necessary to first explain the new extension to the Starcat framework - fuzzy codelets.

Fuzzy Codelets

A fuzzy codelet (FuzzyBehavioralCodelet) exhibits similar behavior to a regular codelet (BehavioralCodelet) with some modifications to how successes or failures are defined. The BehavioralCodelet defines success and failure discretely; the result is one hundred percent either success or failure. The FuzzyBehavioralCodelet provides a more continuous definition; the answer can be partially a success as well as a failure. This allows for a more realistic solution to the Codelet’s environmental tests.

The BehavioralCodelet class defines an object which tests the environment and sends activation to only one collection of Slipnet nodes - the success collection or the failure collection. Each BehavioralCodelet has an associated Success-Failure value which ranges from zero to one hundred. The outcome of the BehavioralCodelet’s environmental tests determines which collection receives this entire value. When the result of the test is false all of the failure recipient nodes receive activation in the amount of the associated Success-Failure value. In this case, the collection of success nodes does not receive any activation. If the test result is found to be true the opposite occurs.

The discrete answer provided by the BehavioralCodelet is disadvantageous to the Botcat system. This can be exemplified with a Codelet that is working to observe obstacles in front of Botcat. There are gray areas in the Codelet’s ability to determine if there is an obstacle in front of it. This gray area lies at the border of an object being an obstacle to Botcat at the present time with the current environmental conditions or not; for this situation, a discrete answer of true or false is not a clear resolution to the problem. To observe if an obstacle is in front Botcat, the Codelet would look one hundred pixels in front of itself. If a wall is seen this would be determined to be an obstacle. If Botcat were one hundred and one pixels away from a wall it would appear that there was no obstacle present since the Codelet is only observing within a range of 100 pixels. It would not be an optimal solution for Botcat to deem that there were no obstacles in front of it, instead, it would be beneficial for Botcat to be aware that there exists a slight obstacle in front of itself. The BehavioralCodelet’s environmental test provides no depth perception in its solution. With the current solution, Botcat would not recognize the difference between a wall that is one hundred and one pixels away compared to one million pixels way. Likewise, moving the wall one pixel closer (one hundred pixels away) results in Botcat being unable to discern whether the obstacle is one pixel or one hundred pixels away. The problem with the discrete solution is solved by using fuzzy codelets.

Fuzzy Codelets provide a better approximation to tests of the conditions present in the battlefield. In reference to the previous example: when asked if there is an obstacle in front of Botcat, fuzzy Codelets allow for a range of answers that could be returned. Possible answers could be that the wall is: partially an obstacle, somewhat of an obstacle, an obstacle, or an unavoidable obstacle. When a wall is one hundred and one pixels away, a fuzzy codelet would judge there is somewhat of an obstacle in front of the robot instead of saying that there is no obstacle at all. These approximate answers are given in values from 0.0 to 1.0. An answer of 1 would be equivalent to true (there is an obstacle) and an answer of 0 would be equivalent to false (there is not an obstacle). This technique results in a range of different answers between the values 0 and 1.

The Fuzzy Obstacle Codelet's fuzzy result sets are shown below in Figure 1. The values on the x-axis are the distance (in pixels) of Botcat from the obstacle in the battlefield, and the value on the y-axis is the percentage of membership in a set. The higher the percentage of membership in a set the more definite an environmental test result becomes. In this example there are two fuzzy sets - Success and Failure. The Success fuzzy set consists of the blue and yellow areas from 0 to 150 pixels. The Failure fuzzy set consists of the red and yellow areas from 50 to 200 pixels. Therefore when the distance to the obstacle is 100 pixels the return value (or membership) for Failure and Success is 0.5, meaning there is to some extent an obstacle and to some extent not. [7] Failure implies that the battlefield does not contain an obstacle in the direction that is being tested. Success implies that there exists an obstacle in the tested direction. The values that lay between the two memberships represent varying levels of objects being an obstacle to Botcat in their current positions at the time of the test.

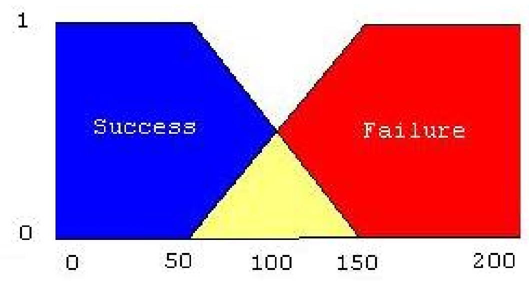


Figure 1.Represents how Fuzzy sets are determined.

In the case of distances in the battlefield that are especially close or far away from Botcat, the fuzzy sets operate as a Boolean value similar to the BehavioralCodelet which does not make use of fuzzy Codelets. For example, if the distance from Botcat to the obstacle is less than 50 pixels, the membership percentage for the Failure set is 0.0 and the success set is 1.0. If the distance is greater than 150 pixels, the membership value for the failure set is 1.0 and the Success set is 0.0. The fuzzy sets have values different than 1 or 0 when the distance is in between 50 to 150 pixels. The rational behind this behavior is that when an obstacle is exceedingly close or far away from Botcat, the decision to declare the presence or absence of an obstacle should be clear-cut. [7]

A FuzzyBehavioralCodelet is composed of the basic Codelet functionality with the additional use of the two fuzzy sets - Success and Failure. As explained above, the BehavioralCodelet, which the FuzzyBehavioralCodelet is based upon, contains an associated membership value that is added to all the Slipnet nodes in one of the two Slipnet node collections (Success and Failure). The FuzzyBehavioralCodelet pairs up the two Slipnet node collections with a fuzzy set. The Success Slipnet node collection is paired up with the Success fuzzy set and the Failure Slipnet node collection is paired up with the Failure fuzzy set. When the FuzzyBehavioralCodelet performs its environment test it calculates the membership values for each of the two fuzzy sets. These membership values (ranging from 0 to 1) are then multiplied by the associated activation value and added to all of the Slipnet nodes in the related Slipnet node collection.

The Codelets use the fuzzy sets by receiving a percentage of the activation values that were applied to their corresponding Slipnet nodes. The BehavioralCodelet adds all the activation value to only one of the two sets while the FuzzyBehavioralCodelet adds a percentage of the activation value to each of the Slipnet nodes in both sets. In this situation, the percentage of activation value that is added is determined by the membership value given to each set. Using the fuzzy sets in Figure 1 as an example, if Botcat's distance was 75 pixels from the obstacle, then 75 percent of the membership value would be added to the Success recipient nodes and 25 percent of the membership value would be added to the Failure recipient nodes. If the distance was 175 pixels, 0 percent of the values would be added to the success recipient nodes and 100 percent would be added to the failure recipient nodes since obstacles at distances that are either incredibly close or far from Botcat lay completely in one of the two sets.

The four Codelets that are used for detecting obstacles are obstacelLeftObserver, obstacleRightObserver, obstacleFrontObserver, and obstacleBackwardObserver. These

Codelets’ names appropriately describe the direction that they are capable of detecting an obstacle in the battlefield. For example, the obstacleFrontObserver Codelet is in the environment continuously checking for obstacles in Botcat’s forward direction. The Codelet tests its specified direction by determining the closest object to Botcat, whether the object is another robot, a wall, or a target. Once an object is found, it will determine the distance (in pixels) from itself to that object. The value of this distance is used to determine if Botcat possesses an obstacle in that direction.

The obstacleFrontObserver Codelet is connected to clearFront and obstacleFront Slipnet nodes. The connection between the Codelet and the Slipnet nodes allows observations made by the Codelets to be sent to the nodes. The Codelet places activation on the clearFront and obstacleFront nodes based upon the assessment of the distance from Botcat to the closest object in the battlefield. For example, if the closest object is determined to be at a comparatively far distance to Botcat, then more activation would be given to the clearFront node over the obstacleFront node.

Genetic Algorithm

A Genetic Algorithm was used to find the best combination of Slipnet node link lengths, node memory weights, and the color of the robot (color of robot was used to differentiate the different generations of robots.) In the process of creating a robot by hand, it was found that there contained over 350 million different combinations of these components. To test each of these combinations it would take over 300 years. Each of these combinations creates different behaviors in the robot. The Genetic Algorithm designed for the Botcat evolution studies works by means of DNA (defined later in this chapter), which encodes all of the different combinations of the three components – node link lengths, memory weights, and robot color. The Genetic Algorithm blends and mutates the Botcat population’s DNA to find a robot that most exemplifies the desired behavior of a robot that will succeed in a Robocode battle. Optimal behaviors that are looked for in a robot are placed in a fitness test. When an individual robot is run against the fitness test, a score is provided based upon how much an individual robot represents those desired behaviors.

The initial Botcat population was created from the handmade robot described earlier in this chapter. The first step in creating the initial population was to encode the handmade robot’s DNA. Once this was done, the DNA was mutated multiple times to create new members to be added to the population. Mutation of the handmade DNA was performed 50 times to start with the desired population size.

Each individual robot in the population is tested against the same fitness test to complete one generation of the Genetic Algorithm. The fitness test will return a number representing how well each individual represented the preferred behaviors (the higher the value the better.) All of the individuals are then sorted by their fitness test score in ascending order. Each individual’s fitness value is then replaced with their individual rank compared to the rest of population. Therefore if the population size were 50 then the highest scoring individual would have an end fitness score of 50 and the lowest scoring individual would have a fitness score of 1.

After all of the individuals in the population are assigned a fitness score, a new generation is created from the mating of the current population. A member of the new generation is created from the mating of two members of the old population. The mating pairs are chosen using a roulette wheel selection procedure. In roulette wheel selection each member of the population is assigned a portion of the roulette wheel that corresponds to their fitness score; the higher the fitness score, the larger the section of the wheel an individual is assigned. The entire wheel size is determined by the sum of all the fitness scores of all the individuals. If the population were of size three, then the wheel would be size 6 (3+2+1). Therefore an individual with a fitness of fifty would have a slice of the roulette wheel of size fifty. Roulette wheel selection gives a preference to individuals with higher fitness scores thereby increasing the chances that more elite members of the population mate and have their DNA propagated on to further generations. [8]

By using individuals’ ranks within their population as their fitness scores instead of the actual scores on the fitness tests, one individual or a small group of individuals are prevented from dominating the roulette selection. If the actual fitness test score were used to rank individuals against each other and one member of the population scored three times higher than the second highest member, this individual would have a much greater chance of being selected for a majority of the mating pairs during the creation of the new population. The majority of the members of the next generation would contain that one individual’s DNA. If most of the population were to possess similar DNA, the diversity of the population would decrease. Low diversity of DNA in the population would decrease the chances of finding new solutions via mating and diversity would only exist through mutation of the DNA.

New generations are created by a combination of both the mating of members of the previous population, and elite members of the previous population being carried over to the next generation. Eighty percent of the new population is created by mating the previous population. The top twenty percent of a population are referred to as the elite; they are carried over from the old population to the new population. This process is done to preserve the top individuals of populations and to not risk that an optimal solution is lost to mutation or mating recombination. There is however a trade off due to the carrying over the elite members of a population, this leads to decreased population diversity. The diversity decreases since generation after generation the same individuals are most likely to mate or carry over their DNA (in tact) into future populations. The more one individual mates with many members of the population the more the following population's DNA resembles that one individual. The positive side of having the elite mate with the population is that it increases the entire population's fitness scores. Allowing the elite to carry over to future populations has the added benefit having further fitness tests run on these individuals to ensure the desired behaviors displayed were not by random chance.

Fitness Test

A fitness test provides a way to measure the degree to which an individual displays the behaviors that are desired. Fitness tests involved decoding each individual's DNA into Botcat and then running Botcat in a Robocode battle. The initial fitness test was designed to test for the desired behaviors of the individual; this was found to be impractical since the initial population did not perform well enough in the tests to receive any scores other than zero. With all the members of the population scoring the same fitness value, the selection algorithm for choosing mates did not work properly. A test plan was then created that contained a string of many tests which were able to slowly evolve the population to display the desired behaviors. A series of fitness tests were created such that one test would lead to the next. The tests were sequenced together to first ensure the robots possessed the ability to move on the battlefield in a coordinated purposeful manner and then to attack other robots in a battle.

The first set of tests involved a dot (target) on the battlefield which when the robot runs over or touches it disappears and reappears in a new location. The robot is scored on the number of targets that it touches during the round time. The robot fitness test consists of many rounds of collecting targets the number collected in all of the rounds are summed together to give the final score of the individual.

Once it is determined that the individuals in the population for the most part performing well, the test was altered to be more difficult. The steps in the evolution of the fitness test were:

- Small battlefield target chaser – Board 400 X 400
- Medium battlefield target chaser – Board 600 X 600
- Large battlefield target chaser – Board 1200 X 1200
- Small battlefield Fight against Non-moving robot – Board 400 X 400
- Small battlefield Fight against multiple Non-moving robots – Board 400 X 400
- Medium battlefield Fight against multiple Non-moving robots – Board 600 X 600
- Large battlefield Fight against multiple Non-moving robots – Board 1200 X 1200

DNA

The DNA of each robot contains three types of information - link lengths, node memory strength, and the color of the robot. Rather than use a bitstring for the DNA, as is done commonly in Genetic Algorithm research, here a list of integers is used to encode the relevant system parameters. Mutation and crossover are accordingly restricted. There are 1,089 links contained in the Slipnet. The Slipnet contains 33 nodes, and is configured in such a way that all nodes are connected to each other using two links. In addition to these links, each node also has a link which connects to itself. This equates to 33 * 33 = 1,089 links. These links are all capable of transferring activation back and forth between each other. The lengths of the Slipnet nodes' links are read from its DNA where they are stored. Link lengths are between 0 and 100; a length of 100 symbolizes that no link exists since the distance between the nodes is too far apart. Activation is transmitted across the link between the two nodes if the distance between the two is a value less than 100.

The second piece of information that the DNA stores is the memory strength of each Slipnet node. The memory strength determines how long a node will hold on to its activation. The memory strength value is between 0 and 100. The higher the value the longer activation takes to dissipate.

The third type of information stored by the DNA is the color of the robot. A robot's body has three sections which can each be colored separately. These sections are the radar, turret, and body. Each color has three values: red, blue, and green. Similar to the values for the link lengths and memory strength, the values for each color of the robot range from 0 to 100. Each of the three body sections of the robot can be a different color. This allows for many different color variations for the robots, and therefore increases the amount of diversity available within the population.

Variation exists in the robot populations via mating and mutation. DNA permits the mating between two DNA objects. This process creates two new DNA objects which each contain a mixture of the original two DNA objects. The process of mutating a DNA object involves randomly selecting link lengths, node memory strength, and the color of the robot. Mutation occurs randomly in the population and randomly acts on various portions of the DNA.

CHAPTER 5

RESULTS

INTRODUCTION

The Botcat experiment was multi-purposed. Prior to performing any portion of the study, it was unknown if the Starcat and Robocode Frameworks would be capable of integration. The success of integration led to the need to manually design a robot that could successfully move within the Robocode environment. It was necessary to create a robot that displayed behaviors which warranted promotion into further experiments. The desired behavior of the hand-made robot was that it interacted with the environment in a manner that could potentially be carried over to succeed in a battle. The chosen robot must exhibit optimal traits since this one individual will prime the initial experimental population.

The robot chosen to prime the Botcat populations was adjusted to exhibit traits which the experimenter deemed optimal. Over the course of mutating and evolving the Botcat populations, many surprising discoveries occurred. The fitness scores of the populations increased greatly from those of the handmade robot's. This result was gratifying however not surprising. The unanticipated results were discovered upon investigating the behavior of the robots which were scoring well on the fitness tests. Some traits were observed which were both abnormal and beneficial at the same time

GENETIC ALGORITHM IMPROVES MANUALLY CREATED ROBOT

A robot was initially created by manually modifying its DNA. The robot chosen for this manipulation process performed at an above average level on the fitness tests. Minimal portions of this robot's DNA were altered. However the affects of this on the robot's behavior were found to be quite drastic. For example, when modifying the memory to recognize an obstacle on the robot's left, if the memory value is increased too much, the robot will constantly turn to the right in an attempt to avoid an obstacle that it believes to be continuously present. In contrast, if the memory value is decreased too much, the robot will run into the obstacle since it did not remember that the obstacle was present. The lengths of the node links within the Slipnet were then modified. Increasing the lengths of some of the links caused the robot to not move at all, and decreasing the lengths caused the robot to constantly move in what seemed like random directions. After numerous manual adjustments coupled with tests, a robot was created that was capable of obtaining an average fitness of 5 in performance test one, meaning the robot obtained an average of 5 targets per test. This was found to be the highest average score that any of the handmade robots possessed.

In order to experiment with evolving a robot that possesses a higher fitness than the most optimal handmade robot, the handmade robot was used to prime an initial experimental population. The population was primed with 50 clones of the handmade robot. These clones each then had 10 percent of their DNA mutated in order to create diversity within the population. Without the presence of diversity it would be impossible for a robot to evolve traits that differ greatly from the initial robot. A plateau would develop in the fitness test scores obtained by the robot population and the Genetic Algorithm would lose its ability to function. To perform the mutation, each Robot's DNA was altered by randomly changing its value from negative 6 to 6. It was found that by mutating the handmade robot's DNA at a greater amount than what was used in the experiment, a population was created which was incapable of scoring any points on the fitness tests.

Each generation consisted of a population of 50 robots; these robots were each tested 100 times in each of three different fitness tests. The average number of targets that each robot obtained in each fitness test was used as the robot's fitness score. The targets in each of the fitness tests were stationary; upon being obtained by a robot, a target would be removed from the battlefield and randomly placed in a new position.

Each of the three fitness tests were similar to each other in all respects except for the size of the battlefield. The first test was a 400 by 400 pixel battlefield, the second was 800 by 800 pixels, and the third was 1000 by 1000 pixels. A selection of the results obtained during the three fitness tests are provided in Table 1. The highest scoring robot in each of 3 generations are listed in the table along with the average number of targets they obtained in 100 runs in each of the three tests. The results of the Handmade robot's runs through the tests are listed as a comparison.

Table 1. Results of Fitness Tests among Three Genetic Algorithm Evolved Botcat Populations

Name	Generation Number	Test 1: $P_{400,400}$	Test 2: $P_{800,800}$	Test 3: $P_{1000,1000}$
Handmade	0	5	2	0
Ghost	500	10	4	1
Pink	1000	15	8	5
Royal	1500	17	12	9

Table 1 lists the highest scoring Genetic Algorithm-evolved Botcat robots from three different generations. The average scores the robots obtained in each of the three fitness tests are listed. The handmade robot is listed in the top row for comparison on the progress made through evolution. Each generation consisted of 50 robots which ran each fitness test 100 times. The results have been rounded off to the nearest whole target number.

In each generation listed in Table 1, each of the robot's scores increased in all three tests. The robot's names are reflections of the robot color (a feature of the DNA); each of the three robots' three body components were all the same color – Ghost was solid white, Pink was a slight pink, and Royal had a purple color. The robots are listed in the table by name so as to acknowledge that the scores listed are those of the individual and not the population. The scores increased greatly from the Handmade robot's to Ghost in generation 500 and from Ghost to Pink in generation 1000. Although the score increased from Pink to Royal in generation 1500, the difference in the scores is not as large as those between the other generations. In all three generations, as well as the Handmade robot, the scores decreased among the individual as the battlefield size increased. The scores for the robots in fitness test 3 are less than fitness test 2, and the scores from fitness test 2 are less than fitness test 1. The generations were not evolved past generation 1500 since the increase in the test scores between this generation and the prior is not high enough to conclude that any major evolution events are likely to occur.

The handmade robot's approach on the battlefield consistently appears to the observer as if each movement is pre-planned prior to execution. When beginning a run in the battlefield, the robot will sit in the same position for a few seconds, and then without any further hesitation, perform a movement towards the target. The time spent without motion gives the impression that the robot is processing its surrounding. If after the movement the robot does not reach the target, it will once again sit motionless for a few seconds and then move towards the target. There were some tests where it seemed as if this robot never recognized the target. When this occurred, the robot would sit as if to process its surroundings, then it would move in a random direction which was not towards the target. Robots in higher generations did not spend much of the time in the battlefield motionless. For the most part the robots were always performing some sort of movement whether it was moving in a certain direction or rotating in place.

UNEXPECTED EVOLVED ROBOT TRAITS

During the evolution of the Botcat robots some beneficial traits evolved which were unexpected. Some robots displayed the behavior during a battle of stopping and/or moving back and forth on a target, this ultimately enabled the robots to successfully obtain the targets. Another beneficial (although initially believed to be odd) behavior which some robots exhibited was spending a portion of the time during the battle ramming into walls (beneficial reasoning explained below). These traits were unexpected as traits the population would evolve; however unknown tactics for battle success in the Robocode environment have made these traits beneficial.

The behavior of stopping or moving back and forth on a target was found to be beneficial due to problems with the recording of a robot’s contact with a target. Once it became evident that a large portion of robots with high fitness test scores were exhibiting this behavior, it was discovered that a portion of the Robocode system had a problem with recognizing that a robot’s position was over a target therefore the robot was not registered as having obtained the target. This piece of the system only checked if the robot was over the target every 10 frames during a battle. It was found that some of the earlier generations’ robots would pass over the target and a score was not registered; this meant that a robot could skim a target or run over a target fast enough for the system to not notice contact being made. Robots which stopped on a target or moved back and forth over it ensured that their presence on the target was long enough for the system to detect their success in obtaining the target.

This trait guaranteed the detection of the robot touching the target thereby leading to an increase in these robots’ fitness scores and an increase in their likelihood to have their DNA propagated into future generations. Robots which did not display this behavior were found to take a lot longer to record a target because they would be found to pass over a target many times without the contact being registered. The time lost by not possessing this trait increased the fitness of individuals that possessed it, thereby propagating it into more individuals in future generations.

The trait of a robot ramming into the walls during a battle possessed another hidden benefit; this again was due to unknown specifications of battle rules in the Robocode framework. The handmade robot tended to not run into walls to a degree that seemed purposeful or beneficial. After evolving this robot and its offspring for 1500 generations it was found that a majority of the higher fitness robots possessed the “ramming” trait. It was initially assumed that the robots’ wall ramming behavior was a mutation-based error that had no major affect on the performance of the robot. After further analysis, a battle feature in the Robocode framework was found which explained why this trait was beneficial.

One of Robocode’s battlefield parameters is the “inactivity time”. During a battle, when a robot performs certain movements it will lose energy points. When a robot loses all of its energy points the battle ends. If both robots were to remain permanently stationary, a battle could potentially continue on indefinitely. The “inactivity time” is a feature of the Robocode framework which is intended to end battles between robots that are not interacting with each other. This is a pre-defined length of time that can elapse during a battle where neither robot has lost ten points of energy. Once this time frame has passed, incremental decay of the robots begins. Incremental decay is when the energy of the robots on the battlefield slowly decays until at least one robot’s energy points reaches zero and the battle ends. The “inactivity time” feature of Robocode was subverted by the evolved Botcat robots’ “ramming” behavior.

When experimentation with the Botcat robots first began, the “inactivity time” was set to an extended length of time; this was a default value in the Robocode system that was unadjusted during creation of the handmade robot. The high “inactivity time” value allowed some robots to exploit the benefits of randomly ramming into walls to prevent the “inactivity time” from elapsing. This behavior increased their time on the battlefield thereby allowing them more time to collect targets. As the time available to collect targets increased, so did these robots’ fitness scores, thus propagating this trait into future populations.

Fuzzy Codelets Increase Botcat's Perception

Crisp codelets were experimented with to test if Botcat benefited from an ability to refine depth perception of objects in its environment due to fuzzy codelets. Crisp codelets are codelets which provide only a discrete solution to environmental tests. For example, when Botcat tests the environment to determine if an obstacle is present, a crisp codelet is only capable of responding with a pass or fail response. A fuzzy codelet can return an answer that provides information as to how far an obstacle exists from Botcat. For example, if an obstacle lies at 60 pixels away from Botcat, since Botcat is capable of testing up to 100 pixels away from itself, the fuzzy codelet would return a result of .4 that the obstacle exists. The fuzzy codelet experiment involved comparing Botcat robots which utilized fuzzy codelets to ones which did not. The procedure involved testing 100 different robots with fuzzy codelets 100 times each in order to obtain an average of the number of targets acquired in each battle. These 100 robots were then run again 100 times with crisp codelets, in order to compare the averages from the two groups.

The test environment consisted of an inactivity time of 500 frames, and a "runaway robot". A "runaway robot" is a non-Botcat robot which has been pre-programmed to run into the corner of the battlefield at the start of the battle. This robot does not interact with or get in the way of the Botcat robot performing the test. By using the "runaway robot" a controlled test environment is created. This robot interacts equally as a component to each robot that is tested, therefore the results of a battle are not altered by its presence and test scores are accurately comparable.

This Table 2 lists the average number of targets obtained by 100 robots during 100 battle runs in each of 4 different fitness tests. The difference between the two averages is listed in the column labeled "Difference". The data listed has been rounded to the nearest whole target number. The battlefield size in each of the 4 tests varied, the pixel size of the battlefield is listed next to each test.

Table 2. Fitness Tests Results from Botcat Populations that were Tested in Various Sized Battlefields. Populations Contained either Fuzzy of Crisp Codelets

Test	Fuzzy Codelet Average Number of Targets Obtained	Crisp Codelet Average Number of Targets Obtained	Difference
1 (400 x 400)	10	9	1
2 (800 x 800)	10	8	2
3 (1000 x 1000)	7	4	3
4 (1500 x 1500)	6	2	4

Four tests were conducted on each of the 100 robots. Test 1 was conducted in a 400 by 400 pixel battlefield, Test 2 - 800 by 800 pixels , Test 3 - 1000 by 1000 pixels, and Test 4 - 1500 by 1500 pixels. The robots used in this experiment were from the 1000th generation from the prior fitness tests. The 1500th generation was not used since it was found that very little difference existed between generation 1000 and generation 1500 in their fitness scores. Generation 1000 was chosen over 1500 because it was assumed that this generation had not yet evolved to its highest potential and the diversity left in the population would allow for the fuzzy codelets to be expressed.

Table 2 presents the results from the fuzzy codelet experiment. Robots that contained fuzzy codelets possessed a higher average test score in each of the four test sets than those that only contained crisp codelets. In general, the average score decreased as the size of the battlefield increased. The fuzzy codelets score on the largest battlefield was on average sixty percent of the score on the smallest battlefield. In comparison, the crisp codelet's score on the largest battlefield was on average twenty-two percent of the score on the smallest battlefield. Future experimentation comparing the two codelets' in various other tests would allow for greater conclusions to be drawn on the effectiveness of fuzzy codelets with this system.

MOVING TARGET

This portion of the experiment was an afterthought which was not considered prior to performing the initial study. The stationary target that was used during each of the experimental tests was a stepping stone meant to develop Botcat's ability to track and fire at opponent robots in battles. The stationary target appears in the battlefield and remains in one place; when the target is obtained by Botcat it is removed from its current location and appears stationary in another location. This continuation of the study replaces the nonmoving target with a moving target to replicate a robot. Again, when Botcat obtains a target, that target is removed from the battlefield and positioned elsewhere.

Table 3 presents the results of the Moving Target study; the average number of targets obtained by 4 sets of 50 robots from 4 different generations are listed. Each generation set of 50 robots were tested 100 times against moving targets at increasingly fast speeds (0 = stationary, 10 = fastest). In this test the initial population used was from the 1000th population of the prior run fitness tests. The rationale for choosing this as the initial population (population 0 in this study) is that the robots have already been evolved enough to have learned how to obtain stationary targets, however they still retain some diversity in the population. This, as opposed to population 1500 which may have hit a plateau in diversity (see conclusion).

Table 3. Fitness Tests Results from Multiple Generations Populations Attempting to Obtain Moving Targets of Various Speeds

Generation Speeds	0	500	1000	1500
0	5	9	11	10
1	2	6	9	9
2	0	3	9	7
3	0	0	5	6
4	0	1	2	1
5	0	0	0	1
6	0	0	1	2
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0

The goal of the tests was for Botcat to obtain as many targets as possible in each battle. Each generation consisted of 50 robots. Each of the 50 robots was tested 100 times at each of the 11 target speeds. Speed 0 is a stationary target and speed 10 is the fastest moving target. The speed increased in equal increments and topped out at speed 10 which was found to be the fastest the Botcat robots were capable of moving. The number of targets obtained by the entire population at each speed was averaged to get the results shown in Table 3. The study concluded at generation 1500; the data collected reveal that the difference in the average number of targets obtained between generation 1000 and 1500 is not great enough to infer that higher generations will become more successful at this test.

The data collected show an increase in the average number of targets obtained by the Botcat populations as the generations increased from generation 0 to 1000. The greatest increase occurred earlier in the generations between generation 0 and 500. From generation 500 to 1000 there was a large increase in the number of targets obtained at the target speeds greater than 0 and 1. Generation 1000 and Generation 1500 obtained a similar number of targets on average, again with the higher generation (1500) scoring a greater number of targets at the increased speeds.

CHAPTER 6

CONCLUSION

The targets used in the Botcat experiment are not features of the Robocode framework, they were added solely for the purpose of the evolution experiments. The targets are primitive placeholders for what would eventually be other robots in the battle. If a competitor robot were to be used in the experiments, it would be necessary for experimental control that the robot was a “sitting duck” (a robot which does not move from the position it is placed in during the entire battle, and does not interact with the Botcat robot) or the “runaway robot”. If this competitor robot were capable of varying behavior during the battles, the results would not be valid. It is imperative that each robot encounter the same environment during each test in order to obtain conclusive results. The target used during the experiments was simple to manipulate from being stationary to traveling at various speed levels.

Certain fitness tests that were conducted involved adjusting the speed of the target to increase the difficulty of the tests. As the test difficulty level increased, the distinction between optimal and less than optimal Botcat robots was discernible. The roulette wheel selection as well as the Genetic Algorithm worked best when there was a large separation in fitness values.

GENETIC ALGORITHM IMPROVES MANUALLY-CREATED ROBOT

It was found that in each generation that is presented in Table 1, as the size of the battlefield decreased, the average score of the robot increased. It can be concluded from this that Botcat performs best in smaller battlefields where the size of the environment is closer to the distance that Botcat is capable of testing. A future test that could stem from this would be to alter both the distance that Botcat is capable of testing in the environment and the size of the battlefield. This experiment would be capable of finding a balance between the two in order to allow Botcat to maximize its performance.

The scores increased greatly from the Handmade robot's to the robot in generation 500 and from that robot to the robot in generation 1000. Although the score increased between the latter two, the difference in the scores is not as large as those between the other generations (Table 1). It was for this reason that the generations were not evolved past generation 1500; the difference in scores was not enough to conclude that any major evolution events were likely to occur in future generations.

UNEXPECTED BENEFICIAL TRAITS

For a behavior to so significantly extend through the population it would either have to have been by chance (which is highly unlikely) or because the behavior was beneficial. As it was most probable that the unexpected behaviors displayed by the robots possessed some adaptable aspects, further research into the Robocode framework was conducted to understand what caused these behaviors to evolve and multiply.

The behavior of stopping or moving back and forth on a target was found to be beneficial due to problems with the recording of a robot's contact with a target. Once it became evident that a large portion of robots with high fitness test scores were exhibiting this behavior, it was discovered that a portion of the Robocode system had a problem with recognizing that a robot's position was over a target therefore the robot was not registered as having obtained the target. This piece of the system only checked if the robot was over the target every 10 frames during a battle. It was found that some of the earlier generations' robots would pass over the target and a score was not registered; this meant that a robot could skim a target or run over a target fast enough for the system to not notice contact being made. Robots which stopped on a target or moved back and forth over it ensured that their presence on the target was long enough for the system to detect their success in obtaining the target.

This trait guaranteed the detection of the robot touching the target thereby leading to an increase in these robots' fitness scores and an increase in their likelihood to have their DNA propagated into future generations. Robots which did not display this behavior were found to take a lot longer to record a target because they would be found to pass over a target many times without the contact being registered. The time lost by not possessing this trait increased the fitness of individuals that possessed it, thereby propagating it into more individuals in future generations.

The trait of a robot ramming into the walls during a battle possessed another hidden benefit; this again was due to unknown specifications of battle rules in the Robocode framework. The handmade robot tended to not run into walls to a degree that seemed purposeful or beneficial. After evolving this robot and its offspring for 1500 generations it was found that a majority of the higher fitness robots possessed the "ramming" trait. It was initially assumed that the robots' wall ramming behavior was a mutation-based error that had no major affect on the performance of the robot. After further analysis, a battle feature in the Robocode framework was found which explained why this trait was beneficial.

One of Robocode's battlefield parameters is the "inactivity time". During a battle, when a robot fires a bullet, is hit by a bullet, or rams into an object it will lose energy points. When a robot loses all of its energy points the battle ends. If both robots were to refrain from actions which decreased their energy points, a battle could potentially continue on indefinitely. The "inactivity time" is a feature of the Robocode framework which is intended to end battles between robots that are not interacting with each other. This is a pre-defined length of time that can elapse during a battle where neither robot has lost ten points of energy. Once this time frame has passed, incremental decay of the robots begins. Incremental decay is when the energy of the robots on the battlefield slowly decays until at least one robot's energy points reaches zero and the battle ends. The "inactivity time" feature of Robocode was subverted by the evolved Botcat robots' "ramming" behavior.

When experimentation with the Botcat robots first began, the "inactivity time" was set to an extended length of time; this was a default value in the Robocode system that was unadjusted during creation of the handmade robot. The high "inactivity time" value allowed some robots to exploit the benefits of randomly ramming into walls to prevent the "inactivity time" from elapsing. This behavior increased their time on the battlefield thereby allowing them more time to collect targets. As the time available to collect targets increased, so did these robots' fitness scores, thus propagating this trait into future populations.

The two traits that were unexpectedly evolved were the ramming behavior and the behavior of moving back and forth over a target in order to ensure it was obtained. Although a byproduct of the system's deficiencies, the latter trait was optimal nonetheless and, along with the ramming behavior, is an important example of the successful evolution that occurred by using the Genetic Algorithm. These behaviors evolved solely due to mutation and selection by the Genetic Algorithm, as the trait was not selected for during the formation of the initial handmade robot which primed the population that displayed these behaviors.

Future experimentation could take place in many facets surrounding these unexpected traits. If an initial generation were tested in the same environment with the same Robocode system configuration, would these traits "unexpectedly" evolve again? If the inactivity time in a battle were decreased (meaning Botcat has less time to remain inactive before the battle is ended) would the ramming behavior evolve faster, or occur in more robots? If the number of frame cycles the system waits before checking if a target is obtained was increased, would the behavior of moving back and forth over a target have become more widespread? Would it have ever evolved at all? These are all questions that could guide future research of the Botcat system.

FUZZY CODELETS

During early experiments that used only crisp codelets (prior to initializing the tests in this study), there was a large amount of gray area that existed in Botcat's tests of the environment. Fuzzy codelets provided Botcat with depth perception, a benefit when it comes to the OAS and TTS. The use of fuzzy codelets resulted in less interference between the OAS and TTS as well as an increase in the number of targets that Botcat was able to obtain on average during the test runs.

Although the differences in scores between the crisp and fuzzy codelets were not incredibly great there was a noticeable difference in the two types of codelet's ability to function in larger battlefields. The fuzzy codelet Botcats' score on the largest battlefield was on average sixty percent of the score on the smallest battlefield. In comparison, the crisp codelet Botcats' score on the largest battlefield was on average twenty-two percent of the score on the smallest battlefield. The ability to more greatly discern an objects distance from itself seems to be an asset that Botcat requires as the battlefield size becomes larger. In smaller sized battlefields (800 by 800 or smaller) the crisp codelets tend to perform at a more comparable level to the fuzzy codelets than they do in larger sized battlefields.

MOVING TARGET

The moving Target portion of the study was conducted in an attempt to adapt Botcat's ability to deal with moving objects in the battlefield. Until this portion of the experiments

Botcat only experienced stationary objects. The moving targets were designed to prepare Botcat for actual battles in the Robocode environment which would consist of both moving competitor robots and crossfire. Botcat would eventually not only need to be able to track another robot in the battlefield in order to fire at it, but it would also need the ability to react appropriately when fired at by competitors. Although the stationary targets used in other experiments in this study worked well to evolve Botcat, they required adjustment if the robot were able to eventually succeed in a battle.

Table 3 presents the results of the Moving Target study. In this test the initial population used was from the 1000th population of the prior run fitness tests. The rationale for choosing this as the initial population (population 0 in this study) is that the robots have already been evolved enough to have learned how to obtain stationary targets, however they still retain some diversity in the population. The results show that that Botcat was able to evolve as the generations increased as long as diversity was maintained within the population.

The data collected (Table 3) reveal that the difference in the average number of targets obtained between generation 1000 and 1500 is not great enough to infer that higher generations will become more successful at the moving target test. For this reason, the experiment concluded at generation 1500. It is most likely that the Botcat populations were not capable of continuing to increase their success at higher generations due to decreased diversity. Measurements of genetic diversity, while potentially useful, can be complex. Here performance-level diversity of the Botcats in the population was considered instead. With few Botcat robots performing successfully in the experiment, the mating pool for the Genetic Algorithm to utilize decreased. As the portion of the population that mated decreased, so did the diversity.

A range of future experiments are possible to attempt to increase Botcat's ability to obtain higher speed moving targets. One of these would be to begin the moving target study with a population from a generation lower than 1000. The lower generation population may not be as advanced in obtaining stationary targets as generation 1000, however it would retain greater diversity which could become a potential asset. A variety of generations could be used to initialize the study's populations. Another option for future moving target experimentation would be to begin the population at generation 0. The population would possess a large amount of diversity and could potentially obtain greater scores than those displayed in Table 3. In order to perform this study it may be necessary to alter the tests in order to evolve Botcat at each speed for a greater period of time.

CHAPTER 7

FUTURE RECOMMENDATIONS

INTRODUCTION

This section provides introductions to future experimentation that can be performed with the Botcat system based upon the previous work in this study. During the course of the initial experiments that were described in preceding sections of this thesis, there were many potential opportunities to branch off into various other experiments. The following sections are extensions to the Botcat studies that could possibly increase the system's ability to become more robust, successful, and display an even greater amount of emergent behavior. Portions of each future experiment were executed to ensure they are feasible, no results from these trials are provided as the extent of research is too minimal to draw conclusions.

DISTANCE DETECTION SENSOR

This experiment will provide the functionality of an additional environmental sensor to Botcat. This sensor will be used to determine the distance either to another robot in the battlefield or the target. The evolution of Botcat with this new sensor will be then tested to determine if the ability to determine distances will enhance Botcat's battle skills.

The Distance Detection sensor will be added by creating a new codelet - targetDistanceDetectorCodelet, and two Slipnet nodes - farDistanceTargetNode and closeDistanceTargetNode. The codelet will find a value for the pixel distance between itself and the target. This value will then be plugged into two fuzzy sets - farDistance and closeDistance; each of these fuzzy sets will be evolved by rearranging and mutating the DNA.

Each time the codelet performs a test of the environment to determine the distance of Botcat to an object in the battlefield, a portion of 100 activation points is given to each of the two Slipnet nodes. The amount of points that each node receives is dependent upon the distance from Botcat to the object. If the object (whether it is a target or another robot) is found to be at a close distance to Botcat then closeDistanceTargetNode will receive a majority of the available points, the same goes for farDistanceTargetNode with an object that is positioned far from Botcat. For example, if the furthest an object can be from Botcat is 500 pixels, and the object is located 50 pixels away, then closeDistanceTargetNode would receive 90 of the available 100 points since the target is located 90 percent of the distance from Botcat to the furthest point. The other 10 points would go to farDistanceTargetNode snce the target is 10 percent away from Botcat.

The two Slipnet nodes' links between each other will be set to a length of 100 (which is the equivalent of not being connected). Botcat will then be run through the Genetic Algorithm to see if the new sensor is incorporated into the system in order to improve its fitness tests. The fitness tests with this sensor will be performed with a battlefield larger than 1000 by 1000. The reasoning for this size requirement is that Botcat currently performs poorly in battlefields of this size and larger, therefore if Botcat becomes more successful in larger battlefields it can be inferred that the improvement is due to the presence of the Distance Detection sensor.

EVOLVABLE FUZZY SETS

This experiment will allow the fuzzy sets in each codelet to possess evolvable parameters. In the current code there exists a codelet which determines if an obstacle is present in front of the Botcat's current position. Other robots in the battlefield as well as walls that block Botcat in any direction are considered obstacles. (At any position on the battlefield there is always a wall present in front of Botcat, therefore the concept of a wall being an obstacle is relative. If a wall is one thousand pixels in front of Botcat it may not be considered an obstacle, but if it was one hundred pixels away it might be an obstacle. Estimates must be made as to what distances would and wouldn't be close enough to be considered an obstacle. These estimated groupings of distances would initially be arbitrarily determined and therefore may not be optimal. By providing the capability for these distance groupings to be read from Botcat's DNA, different estimates of the groupings could be tested in order to evolve a more optimal Botcat system.

By allowing two values in each fuzzy set to be varied, Botcat's DNA could adjust the different distance groupings which are used to determine whether a wall is an obstacle or not. The current code uses two fuzzy sets to determine if a wall's distance is or is not considered an obstacle. Each fuzzy set's membership function is determined by two variable values. As in earlier studies within this thesis, the fuzzy sets used in this experiment are known as the Failure Set and the Success Set. These two sets are typically situated such that one covers the lower portion of distance values and the other set covers the upper portion of distance values.

Figure 2 (below) is provided again from an earlier section for reference purposes. In the Success Set, if the distance value is less than 50 then the membership of an object in this set is 1, meaning the object is definitely an obstacle at its current position. If the distance value of an object from Botcat is greater than 150 the object's membership in the Success Set is 0, meaning the object is not an obstacle to Botcat in its current position (Figure 2). The same is true for determining membership of an object in the Failure Set, where an object's membership in the set is 0 at distances less than 50 and 1 at distances greater than 150. When the distance value is between 50 and 150 the line function between points (50, 1) and (150, 0) in Figure 2 is used. The values 50 and 150 are the two points which may be varied in the fuzzy sets' DNA in order to find the most optimal grouping of the distances as to what is and is not considered an obstacle.

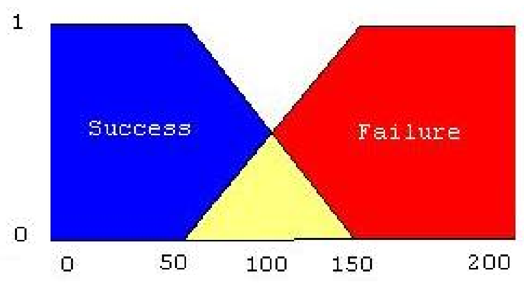


Figure 2.Represents how Fuzzy sets are determined.

The portion of this study which involves running fitness tests would be similar to the Moving Target study described in section 6. In the moving target study 100 member populations over the course of many generations were tested against an increasingly fast target. The robots were run through 100 battles each to determine the average fitness score which was then used for mating selection with the Genetic Algorithm. Likewise 100 member populations could each be tested with varying distance groupings to determine membership in the fuzzy sets. Each member in the population would run 100 battles with each variation of the grouping estimates and the average fitness score would be used by the Genetic Algorithm to determine mating.

REGRESSIVE ROBOT

As the name implies, the Regressive Robot Experiment would remove the advanced codelet logic the currently exists in Botcat in order to create a more primitive robot. The codelets currently present in Botcat are the following: obstacleForwardCodelet, obstacleBackwardCodelet, obstacleLeftCodelet, and obstacleRightCodelet. These codelets perform complicated geometry and algebra to determine if there is an obstacle in their specified direction. The number of Slipnet node layers would be increased to substitute for the removal of the advanced codelets.

The Slipnet nodes that are already present - obstacleRight and clearRight, currently get their activation directly from observations. This process of receiving activation would be replaced by a new level of Slipnet nodes which would have links to the nodes listed above. The new Slipnet level would contain Slipnet nodes and codelets which would detect information in the battlefield environment; these detected values will then be used to calculate the direction of obstacles. The codelets that perform this operation will be evolved through the Genetic Algorithm to perform optimally. By mapping raw inputs from the robot’s environment directly to codelets and Slipnent nodes, new and unforeseen behavior can emerge. This experiment will bring a whole new level to creating an autonomous robot. The existing Botcat system was provided with the ability to perform complicated mathematics; the removal of this will require the system to evolve the ability to logically analyze the environment. Botcat’s emergent behavior that arises from this process will be less linked to the applications which the system was initially provided.

FUTURE BOTCAT FEATURES

The following list contains suggestions for new sensors which would extend Botcat’s current Slipnet nodes. The experiments which would be performed with the presence of these sensors will potentially lay the foundation for new behavior to arise.

- Distance from Botcat to a target within the battlefield environment.
- Ability for Botcat to fire at competitor robots during a battle.
- Track Botcat’s mood.

- Ability to track other robots within the battlefield environment.
- Ability to track robots' turrets within the battlefield environment.

SHORT TERM MEMORY: DYNAMIC SLIPNET NODES

The Short Term Memory experiment requires the Botcat system to contain an area in the Slipnet which contains Slipnet nodes that are created dynamically during a battle run. In the region where these dynamic Slipnet nodes are housed, the links produced between the newly created nodes would be dictated by the DNA. When DNA mutation between the generations' populations occurs, the links between these nodes would become altered. An example of a dynamically created Slipnet node would be a node that is created whenever Botcat sees another robot in the battlefield; the Slipnet node created would be specific for that robot. This Slipnet node would only exist for the presence of that individual robot, when the robot ceases to exist so will the node. This functionality will simulate the presence of short term memory in Botcat. This experiment would coincide with the suggested features of Robot and Turret tracking as listed in the "Future Botcat Features" section above.

BOTCAT COMPETING FOR RESOURCES

This experiment will study the co-evolution of a population of Starcat robots (Botcats) while they are each competing against each other for the same resources. This study would model biological evolution, where species have to compete with each other for resources and mates in order to spread their genes. The current Botcat experiments in the research performed for this thesis do have competition between individuals for mates, however not for resources. Competition between mates occurs due to roulette wheel selection in the Genetic Algorithm; this type of selection gives individuals with a higher fitness a better chance of being selected for mating.

In the experimentation for this thesis, each individual within the population was tested in the environment without any competition existing with other individuals for resources. This fitness test involved one Starcat robot moving around the battlefield in order to obtain as many targets as possible. Each time the robot touches a target a new target is created and randomly placed on the battlefield. The time limit is the only pressure that the robot faces while trying to collect as many targets as possible.

In this suggestion for future experimentation involving competition for resources, would place two or more individuals from the population into the battlefield environment at the same time. Here, there would be multiple individuals competing for the same resources (targets). The fitness test would consist of multiple Starcat robots on the battlefield competing to obtain as many targets as possible before the other robots get them. Only one target at a time would be available for the robots to obtain. As is the case in the original experiment, once a target is acquired by a robot, the target would disappear and another one would be placed randomly on the battlefield.

The competition between robots may help to evolve the population faster. With individuals in direct competition with each other, new techniques would need to evolve in order to successfully compete for limited resources. By having more members of the population being tested in parallel, the amount of time spent finding each individual's fitness would decrease, thereby allowing the study to extend through an increased number of generations. This experiment will potentially allow Botcat to not only evolve at an accelerated rate, but to also evolve for an extended period of time, both of which could produce results that have not been observed in prior studies.

BOTCAT POPULATION CO-EVOLUTION

This experiment would evolve two or more separate Botcat populations separately from each other under identical environmental conditions. The populations would not have contact with each other; they will evolve independently with no inter-population mating occurring. The environments in which the Botcat populations are tested would be controlled to ensure each population encounters identical environmental conditions. Having multiple populations in which individuals within each population are only capable of mating with individuals in the same population simulates the separation of species in the real world.

Fitness tests would involve placing one or more individuals from each of the two or more populations onto the battlefield to battle for targets. There would be inter-population battles. Each individual's fitness score would be the total number of targets that they obtained during the battle. For example, let there exist two separate populations (pop1 and pop2), and an individual from each population (agent1 and agent2) is placed into the same battlefield together. Assume that after a battle, agent1 obtains 3 targets and agent2 obtains

12. It cannot be inferred from these scores that since agent1 scored poorly compared to agent2 that it possesses a low fitness compared to its population nor can it be inferred that agent2 has a high fitness compared to its population. Each individual is only compared to its own population and not to the robot which it competed against. If pop1 has an average fitness of 2, agent1 would be considered to have a high fitness score and would therefore have an increased probability of being selected for mating. If pop2 has an average fitness of 20, agent2, with a fitness score of 12, would have a lesser probability of being selected for mating.

Some patterns in real world species that could potentially result from this experiment are annihilation/dominance and predator/prey. The annihilation and dominance could come about if one population evolves more greatly than the other(s). The example above was leading towards an annihilation and dominance relationship, where pop1's average fitness was 2 and pop2's average fitness was 20. The difference in fitness scores among individuals in a battle do not affect the status of an individual within its population, however the fact that an individual is battling another individual with a much higher fitness score does decrease the likelihood of performing well and obtaining a high fitness score. By battling a population with a comparably large average fitness score, pop1's fitness could become lower. If pop1's fitness becomes so low that all the fitness scores are 0 then the population is annihilated. This annihilation is due to the roulette wheel selection that occurs in the Genetic Algorithm. The Genetic Algorithm loses its ability to select one individual over another for mating since the population does not possess varying fitness scores. In this case population pop1 would be choosing individuals at random for mating; the odds of producing successful offspring by chance are very unlikely.

REFERENCES

[1] Lewis, J., Lawson, J., *Starcat: An Architecture for Autonomous Adaptive Behavior* in Proceedings of the 2004 Hawaii International Conference on Computer Science, pp. 537-541. January 15-18, 2004. Honolulu, HI. 2004..

[2] Mitchell, M., Analogy-Making as a Complex Adaptive System In L. Segel and I. Cohen (editors), Design Principles for the Immune System and OtherDistributed Autonomous Systems. New York: Oxford University Press, 2001

[3] Lewis, J. *Emergent Representation and Adaptive Behavior Using the Starcat Framework*. To appear in *Complexity*.

[4] Lewis, J., Streit, S., (2006) "Spatial Mapping and Navigation in the Madcat Architecture" in *Abstracts of the 2006 International Conference on Cognitive and Neural Systems.* May 17-20, 2006. Boston, MA

[5] Newell, A., Simon, H. 1963 in *Russell & Norvig* 2003, pp. 18.

[6] Claiborne, A., Fricke, M., Lopes, L., Lewis, J., Luger, G., *Emergent Representation in a Robot Control Architecture*. Feb. 8, 2000.

[7] Negnevitsky, Michael., *Artificial Intelligence: A Guide to Intelligent Systems*. Addison Wesley 2005

[8] Crutchfield, J., Das, R., Mitchell, M., *Evolving Cellular Automation with Genetic Algorithms: Review of Recent Work*. Processing of First International Conference on Evolutionary Computation and Its Applications, Moscow. Russian Academy of Science 1996.

APPENDIX A

GLOSSARY

Activation: is transferred to slipnet nodes when the concept that is specific to the slipnet node is recognized in the environment. When a slipnet node's activation value is high enough, codelets will be produced and sent into the environment.

Artificial Intelligence (AI): is the study of intelligent behavior in non-living objects, both the existence of intelligence and the goal of creating machines that are capable of expressing it.

Battlefield: is the GUI environment that the Robocode robots, including Botcat, battle in. This is the environment that the Botcat study utilized.

BotCat: This is a reference to any Robocode robot which uses the Starcat framework.

Complex Adaptive System (CAS): present the idea of Emergent Behavior being produced from machines. These systems consist of many intricate components which work together to produce the system's behavior.

Codelet: is a collection of commands from the Slipnet which are to be performed on objects in the Workspace.

Concept: An idea that a Slipnet node represents, such as the concept of "backwards" or "left".

Copycat: Copycat is one of the first analogy-making programs; it was created by Douglas Hofstadter and Melanie Mitchell. Copycat's domain is specific to the micro-world of letters of the English alphabet and their relationship to each other. The system uses the same structures as the Starcat framework.

DNA: The DNA of each robot contains three types of information - link lengths, node memory strength, and the color of the robot. The Genetic Algorithm blends and mutates the Botcat population's DNA in order to create members of the next generation's population.

Emergent Behavior: is behavior that is displayed or produced by a system that is not predefined or programmed into the system from an outside source.

Expert System: a system that is capable of providing information on specific topics. Each Expert System is designed specifically for one domain. The system utilizes information that is obtained from many experts in the field of the system's domain subject matter.

Failure Set: The set in which the membership points from a fuzzy codelet's environmental test result will be awarded to given the test returns a failure. For example, if the fuzzy codelets test to see if an obstacle exists in the Battlefield, and the result is that no obstacles are present, then the Failure Set will receive most of the membership points. The greater the amount of membership points in the Failure Set, the more likely it is the case that no obstacles exist in the Battlefield.

Fitness Score: The score awarded to an object (or population) based upon its performance in a test. The object and test can vary, and Fitness test scores are not comparable between different types of tests or objects. For example, one Fitness Score could be based upon the number of stationary targets that an individual robot was able to obtain in 5 rounds, and another Fitness Score could be based upon the population's ability to obtain moving targets in 100 rounds; these Fitness Scores would not be comparable to each other.

Fitness Test: Any assessment that involves testing the performance of an object (or group of objects).

Fuzzy Codelet: A codelet that is capable of providing a non-discrete answer to an environmental test. For example, a Crisp Codelet may respond to whether an obstacle is present with a response of 0 meaning no obstacle is present, or 1, and obstacle is present; a Fuzzy Codelet is able to be more specific with a response such as .8, and an obstacle is 80 percent likely to be present.

Fuzzy Set:

Generation: A robot population which is genetically distinct from predecessor or succeeding populations due to mating and DNA mutation activities. A given generation is created based upon its predecessor generation, and is used to give rise to the succeeding generation.

Genetic Algorithm: The Genetic Algorithm blends and mutates the Botcat population's DNA to find a robot that most exemplifies the desired behavior of a robot that will succeed in a Robocode battle. The Genetic Algorithm is used to find the best combination of Slipnet node link lengths, node memory weights, and the color of the robot.

Inactivity Time: a feature of the Robocode framework which is intended to end battles between robots that are not interacting with each other. This is a pre-defined length of time that can elapse during a battle where neither robot has lost X points of energy. Once this time frame has passed, incremental decay of the robots begins.

Java: A programming language from Sun Microsystems.

Link: Links are used to connect slipnet nodes in the Slipnet in order for activation points to be transferred between them. Links can vary in size; this size is one of the factors that the Genetic Algorithm uses to evolve the Botcat population.

Madcat: An autonomous robot system created by Dr. Joseph Lewis and colleagues at the University of New Mexico which utilizes the Starcat Framework. A unique feature of the Madcat system is mapnet; this is where environmental features that are consistently observed in the snapshots are stored.

Mating: Recombination of the Botcat DNA between two robots from one generation in order to produce a robot for the succeeding generation.

Node: Nodes reside in the Slipnet, and are a way that Starcat represents concepts.

Observer: A type of codelet that observes aspects of the environment.

Obstacle Avoidance System (OAS): The OAS's basis collects the inputs of where obstacles in the battlefield are located relative to the robot. This system works to ensure Botcat avoids obstacles.

Obstacle: Any object in the Battlefield environment that Botcat can come in contact with; walls as well as other robots are considered obstacles.

Physical Symbol System Hypothesis (PSSH): A theory by Allen Newell and Herbert Simon, the PSSH states that "A physical symbol system has the necessary and sufficient means of general intelligent action" [5]. A physical symbol system is defined as being a system that manipulates individual symbols into varying patterns so that meaning can be drawn from the combined formation of these symbols.

Population: A set of robots which are of the same generation and have been evolved to the same degree.

Ramming Behavior: This behavior involves a robot repeatedly running into a wall to ensure the inactivity time of the battle does not elapse.

Representation: Representation is the concept of how knowledge is stored or represented by a system.

Robocode: an open source framework which was created by Mathew Nelson at IBM. Code is written for Robocode in order to control robots in an environment where they are able to interact with each other.

Roulette Wheel Selection: In roulette wheel selection each member of the population is assigned a portion of the roulette wheel that corresponds to their fitness score; the higher the fitness score, the larger the section of the wheel an individual is assigned. The entire wheel size is determined by the sum of all the fitness scores of all the individuals. The Genetic Algorithm uses this type of selection to evolve the Botcat population

Runaway Robot: An altered form of a competitor robot from Robocode. The Runaway Robot will run into a corner of the battlefield at the beginning of a battle and remain stationary there to ensure that it does not interfere with the Botcat fitness tests.

Slipnet: is a collection of connected nodes which represent concepts.

Starcat: Starcat is a framework which was created by Dr. Joseph Lewis to provide the basic functionality of the Copycat program; it allows other domains to utilize the essential functionality of Copycat. The Starcat framework is not specific to any domain; it allows custom domain-specific modules to be added on top of the main Copycat engines.

Success Set: The set in which the membership points from a fuzzy codelet's environmental test result will be awarded to given the test returns a success. For example, if the fuzzy codelets test to see if an obstacle exists in the Battlefield, and the result is that there is an obstacle present, then the Success Set will receive most of the membership points. The greater the amount of membership points in the Success Set, the more likely it is the case that an obstacle exists in the Battlefield.

Target: An object in the battlefield that Botcat strives to obtain. Targets can be stationary or moving.

Target Tracking System (TTS): The TTS's basis collects inputs as to where the target is positioned relative to the robot. The TTS works in order for Botcat to obtain Targets in the battlefield.

User Interface (UI): A method for the user to interact with the system/program.

Workspace: the environment where objects are observed and analyzed.

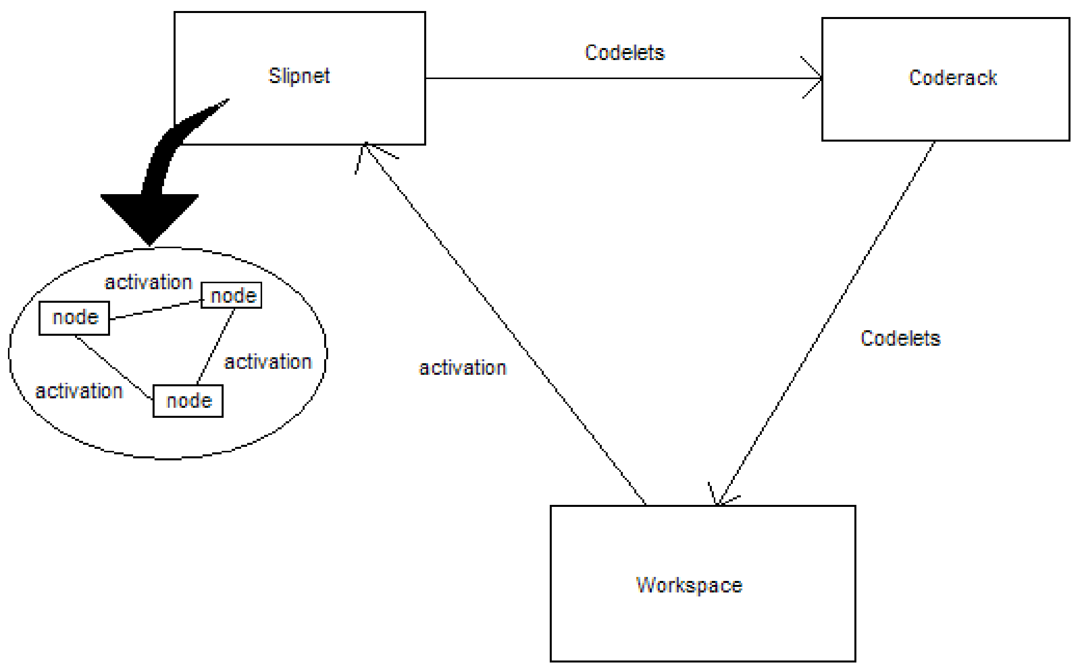
APPENDIX B

ACRONYMS

AI	Artificial Intelligence
CAS	Complex Adaptive Systems
GUI	Graphical User Interface
OAS	Obstacle Avoidance System
PSSH	Physical Symbol System Hypothesis
TTS	Target Tracking System
UI	User Interface

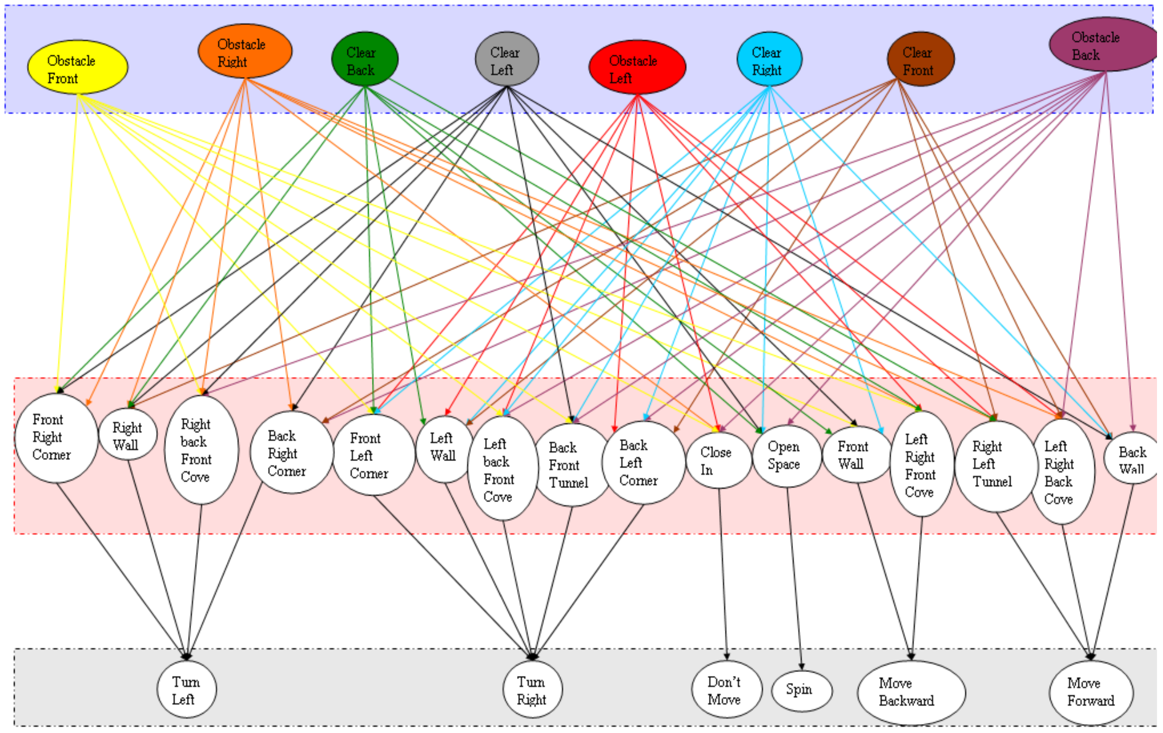
APPENDIX C

STARCAT QUICK REFERENCE


Slipnet
Codelets
Coderack
activation
node
node
activation
activation
node
activation
Codelets
Workspace

APPENDIX D

HANDMADE BOTCAT SLIPNET


Obstacle Front
Obstacle Right
Clear Back
Clear Left
Obstacle Left
Clear Right
Clear Front
Obstacle Back
Front Right Corner
Right Wall
Right back Front Cove
Back Right Corner
Front Left Corner
Left Wall
Left back Front Cove
Back Front Tunnel
Back Left Corner
Close In
Open Space
Front Wall
Left Right Front Cove
Right Left Tunnel
Left Right Back Cove
Back Wall
Turn Left
Turn Right
Don't Move
Spin
Move Backward
Move Forward

ABSTRACT OF THE THESIS

Starcat and Robocode Genetic Algorithm
by
Lance W Finfrock
Master of Science in Computer Science
San Diego State University, 2008

Starcat is a framework which provides the common functionality of Copycat, an autonomous analogy-making system. This functionality is capable of integration into many domains. This paper describes an experiment which integrated the Starcat framework into Robocode, a program that allows virtual user-modified robots to compete against each other in battles. The integration of these two frameworks resulted in an autonomous robot known as Botcat, which was used to prime a population of robots.

The Botcat robots were run through fitness tests in the Robocode environment, and then evolved using a Genetic Algorithm with roulette wheel selection. Many generations of Botcat robots were created using this technique in order to obtain optimal individuals which had autonomously learned how to excel in the Robocode environment. Emergent behavior was displayed from the robot populations, and some unexpected results were discovered. It was concluded that the as the generations increased, the populations tended to perform better in the tests; however at the exceptionally higher generations the diversity in the population tapered and the populations' ability to evolve beneficial traits decreased. Various extensions of the study included comparisons involving fuzzy and crisp codelets, as well as alterations of the fitness tests to produce a more advanced Botcat individual. The results of the study are provided as well as suggestions for future experimentation with the Botcat system.