Dynamic Strategy Generation in computer games using Artificial Immune Systems

by

John Slocket

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Master of Science
in
Computer Science

Guelph, Ontario, Canada

© John Slocket, August, 2011
ABSTRACT

DYNAMIC STRATEGY GENERATION IN COMPUTER GAMES USING ARTIFICIAL IMMUNE SYSTEMS

John Slocot  
University of Guelph, 2011

Advisor:  
Professor S. Kremer

This thesis investigates the use of an Artificial Immune System as a method for dynamically creating computer game strategies in a non-deterministic environment.
Statement

In modern computer games there are many opportunities for machine learning and artificial intelligence to integrate. One section that has not embraced contemporary machine learning techniques is the area of computer controlled characters. The predominant way for the behaviour of these characters to be controlled is through the use of scripts, or prescribed responses to events generated during the course of the game.

Artificial Immune Systems are usually not considered among the mainstream constituents of contemporary machine learning. In this thesis, it is shown that Artificial Immune Systems can be used as a practical solution that bridges traditional methods of controlling a computer game character through scripting and a more contemporary approach. An implementation of an AIS that is appropriate for use in computer games is presented with the method for dynamically generating strategies. Also it is shown that by altering the parameters of the AIS we can change the performance and behaviour of the character controlled by the AIS.
Acknowledgements

My advisor Professor Kremer for all his hard work and good advice.

My wife, Emer, for having patience with me during the creation of this document.
Contents

ABSTRACT ........................................................................................................ iii
Statement .......................................................................................................... iv
Acknowledgements ......................................................................................... v
List of Tables .................................................................................................. vii
List of Figures ................................................................................................ vi
Introduction ..................................................................................................... 1
History of AI in computer games .................................................................... 1
Chapter 1 - Adaptability of opponents in modern computer games ............... 4
  Current techniques for controlling character behaviour in computer games .... 6
  The quest for a more engaging opponent ..................................................... 8
  Adaptation of opponents in video games ................................................... 10
  Offline learning and online adaptation ....................................................... 13
  Unpredictability and the illusion of intelligence ......................................... 14
  Conclusion ................................................................................................... 15
Chapter 2 - Artificial Immune Systems .......................................................... 16
  The Biological Immune System .................................................................. 16
  The Immune Response ............................................................................... 18
  The Artificial Immune System (AIS) ......................................................... 18
  Components of the B Cell model ............................................................... 26
  Artificial Immune System for controlling video game opponents .............. 30
  Starting with a pre-populated current selection of successful strategies ...... 32
  Conclusion .................................................................................................. 33
Chapter 3 - Artificial Immune Systems for strategy generation ................. 34
  The structure of the AIS ............................................................................ 34
  AIS initial implementation .......................................................................... 37
  Structure of the AIS algorithm ................................................................... 38
  Robocode Project ....................................................................................... 40
  Evolution within the Robocode Environment .......................................... 41
  The Testing Procedure ............................................................................... 44
  Evaluation of AIS strategies ....................................................................... 45
  Mutation of AIS strategies ......................................................................... 46
Conclusion ............................................................................................................................................. 47
Chapter 4 - Testing AIS adaptation in a game environment ................................................................. 48
Evaluation of changes in parameters of the AIS .................................................................................. 48
Change in Population Size .................................................................................................................. 50
Change in Maturation Factor ............................................................................................................. 52
Change in Mutation Rate .................................................................................................................... 55
Change in Maximum Strategy Length ............................................................................................... 58
Conclusion ............................................................................................................................................. 60
Chapter 5 - How the AIS recalls previous encounters ........................................................................ 61
How the AIS remembers past encounters ......................................................................................... 61
Time taken to adapt to changing opponent strategies ....................................................................... 64
Factors that can influence the recollection of successful strategies ................................................ 68
Conclusion ............................................................................................................................................. 70
Chapter 6 - Analysis of the test results ............................................................................................... 72
Evaluation of the results ..................................................................................................................... 72
Control over the generated results .................................................................................................... 75
AIS Implementation Evaluation .......................................................................................................... 76
Robocode as a platform for adaptation and automated testing .......................................................... 78
Conclusion ............................................................................................................................................. 79
Chapter 7 - Conclusion and further work ........................................................................................ 81
AIS as a suitable vehicle for learning and adaptation in games ......................................................... 81
Offline learning .................................................................................................................................... 83
Distributed learning .............................................................................................................................. 84
Further work .......................................................................................................................................... 85
Appendix A ........................................................................................................................................... 94
Appendix B ........................................................................................................................................... 95
**List of Tables**

Table 1 Comparing two strings from a library composed of {A .. Z} where the same character in the same position increases the affinity by 1, for a total of 4 in this case ...28

Table 2 Selection of individual genes from 10 libraries to make up a representation of an antigen in the AIS ..........................................................29

Table 3 Actions for tactic A ..................................................................41

Table 4 Actions that will be combined to produce strategy ABA ..................43

Table 5 Final actions that make up strategy ABA .......................................43

Table 6 Parameters for the Population Size test .......................................51

Table 7: The average affinity from each of the ten generations run by the AIS on each population size ........................................................................51

Table 8 Compiled results from the Population Size tests ..........................52

Table 9 Examples of some average affinity calculations ............................53

Table 10 Parameters for the Maturation Factor test ...................................53

Table 11 The average affinity from each of the 10 generations run by the AIS on a maturation factor of 0.1 to 0.9 ..................................................54

Table 12 Compiled results from the Maturation Factor tests .....................55

Table 13 Parameters for the Mutation Rate test .......................................56

Table 14 The average affinity from each of the 10 generations run by the AIS on a Mutation Rate of 0.1 to 0.9 ..................................................57

Table 15 Compiled results from the Mutation Rate tests ..........................58

Table 16 Parameters for the Strategy Length test .....................................58

Table 17 The average affinity from each of the 10 generations run by the AIS on a strategy length of 1,3,5,7 and 9 ..................................................59

Table 18 Compiled results from the Strategy Length tests ........................60

Table 19 Parameters for the Offline Learning and subsequent Adaptation ....63
Table 20 The top ten strategies from each of the strategies encountered ordered by effectiveness .................................................................64
Table 21 Adaptation of the Toorkild strategy when the strategy has been previously encountered against when it has not been previously encountered ........................................65
Table 22 Results of the amount of generations taken to reach the average affinity........68
Table 23 The results generated against the Toorkild robot after 10 generations, with a mutation rate of 0.2, maturation factor of 0.1 and a max strategy length of 10. From the results, Strategy G is the most successful. .................................................................75
List of Figures

Figure 1 Antigens and how the Antibody recognises them........................................17
Figure 2 The components of the AIS showing within what realm each of the components lie .........................................................................................................................36
Figure 3 The AIS process for the generation of a collection of the current strategies that have been selected to compete against the current opponent strategy.......................38
Figure 4 The average affinity recorded when training against multiple strategies.
Generations 1-10 against Toorkild, Generations 11-20 against Thorn and Generations 21 - 30 against WaveShark ........................................................................................................63
Figure 5 The first encounter of the AIS against the Thorn strategy .........................67
Figure 6 The second encounter of the AIS against the Thorn strategy......................67
Figure 7 The mutation process as implemented in the Robocode AIS during Affinity Maturation for the strategy ABCDEFG. Low affinity produces more mutation than high affinity.....................................................................................................................76
Figure 8 and AIS can share it's strategies with other instances of the same game. ....85
Figure 9 A sample Robocode program used to control a Robocode robot from Eisenstein's TableRex program........................................................................................................86
Introduction
In modern computer games, the most common way of controlling computer characters' actions and responses is through scripting, or a set of rules that make up the computer characters' behaviour. While this has proven to be a versatile and robust way of controlling behaviour, there are inherent limitations to this technique, most notably the ability to act outside the scripted behaviour or adapt to changes within the game environment.

This thesis looks at an alternative approach for controlling computer characters behaviour, in the form of a dynamic strategy generation system. This system is based on an implementation of an Artificial Immune System, a computer construct based on the biological immune system. We will try to demonstrate that such a system provides a way to control computer characters, and also allows for the characters to learn and adapt when faced with changes in the opposing player’s strategies.

The Robocode (Robocode Project, 2001) environment will be the basis for the evaluation and testing of engagements between AIS enabled agents and statically scripted agents. Robocode provides a mature platform where various different computer controlled agents can be pitted against each other, and also an excellent architecture for scripting and dynamically changing the agents behaviour.

History of AI in computer games
Since computer games have been in existence there has been some form of Artificial Intelligence responsible for controlling the computers interaction with the human player. Almost all computer games have some sort of decision making process within them. At
the top level is the process for which the overall game behaviour is defined, i.e. what is
the game trying to achieve, or how is the game designed to be challenging for the human
player. A simple game like PONG (Wikipedia - Pong) might have two goals:

To stop the ball getting past the paddle

Return the ball in the direction where the human player is least likely to intercept it.

A third, less obvious goal, might be to allow the human player some level of success if
they are playing well against the computer. This way there is some entertainment and
achievement value to the game.

In games of this sort the human player competes directly with the computer opponent.
Most computer games are of this nature, where the game has some intrinsic goal and has
been programmed to achieve that goal while preventing the human player for achieving it
first. It would not be difficult to program the PONG game to never let the ball get past or
to always calculate the most difficult angle of return for the human player. If this was the
case it would ultimately defeat the point of the game, and that is to present the human
player with a challenging but not impossible task. The player needs to feel like they have
a chance to win; otherwise playing the game would be futile. The overlying objective of
any computer game should be to entertain the end user, whether it is by designing the
game to be entertaining, or by the entertainment being generated from the challenge
pertaining to the game.

As games evolved, so did the complexity of the in-game characters. Some characters
would be hostile or friendly based on the human player’s statistics, alignment or previous
actions (Alt & King, 2002). Some of these games were based around role-playing, where
the human player's character would interact with the computer characters. These computer characters would respond to the human input, as the human player would pick some statement or action and the character would respond with the appropriate response. Scripting also emerged as a convenient and appropriate way of controlling the interaction of computer controlled characters within a game environment. Game designers, who are not directly involved with the development of the game, now have the ability to define and control the behaviour of the game's characters. This lead to more complex and engaging experiences, where the game characters could almost seem to be imbued with emotion and intelligence.

Today's games have the potential to be so immersive that it is now possible for people to spend the majority of their time within game environments. While this is not recommended, it is a testament to the capabilities of modern computer games to capture and hold the interest of players so much so that it becomes the main motivation in their lives.
Chapter 1 - Adaptability of opponents in modern computer games

This chapter explains the concept of non-player characters and computer-controlled agents. Also, we describe how the interaction between the human player and the computer characters has become more complex as computer games have become more immersive and interactive. We describe current techniques for controlling computer character behaviours in different environments and talk about how the response to the human player's actions can be improved upon. We talk about opponent modelling and current implementations of machine learning in computer games.

The method employed by most modern computer games for controlling character's behaviour within the game is through scripts (Berger, 2002). This approach has been predominant over the last decade and has shown little deviation in its implementation. The use of scripts for character AI is an attractive solution as it separates the character's behaviour from the game engine and lets designers have complete control of the implementation. It also provides the ability for the game to be changed and modified outside of the game engine in a way that limits the introduction of bugs, and is generally in a format that can be understood by non-programmers. The designer can interrogate what actions that a certain computer character has been assigned. We refer to this as observability, and is very important for allowing games developers to see the behaviour of the game character as it goes through its designated actions.

Another characteristic of AI scripting is that the behaviour of the scripted character can very quickly be evaluated and determined if the script works as expected. There will be no unexpected deviations unless the script dictates it. It is this level of control that
enables the designers to have the characters behave exactly as they envisioned. We refer to this as *control*.

While AI scripting offers a very controlled way of managing behaviour, it is traditionally incapable of learning or adaptation. One major pitfall is that scripts intrinsically lead to predictable character behaviour. The scripted characters will follow their script regardless of anything that the player may do or how the game may change. Techniques such as time delays and randomization of variables are often used to add unpredictability to the behaviours of the characters. There is some work into generating these scripts dynamically, with results that show this technique is promising (Spronck, Sprinkhuizen-Kuyper, & Postma, 2003).

The goals of adapting to the current player's strategies or techniques is a challenge that lies outside of the realm of AI scripting. The *adaptability* of computer characters is typically a process which requires some amount of learning, searching and randomization. With advances in soft computing techniques and computer processing power, a question that has been raised over and over is to why we cannot employ these learning techniques to computer controlled characters. Some attempts to address this have been implemented using modern soft computing techniques such as neural networks and genetic algorithms. These attempts have not been embraced by mainstream game development, the main reasons being the lack of *control* to modify the output behaviour and lack of *observability* on how the modified behaviour is displayed (Manslow, 2002).

Some of the issues with modern soft computing techniques in terms of learning and adaptation, is that they can be unpredictable, computationally intensive and difficult to
implement. In the context of game design, time and resources are precious commodities, and anything unknown may cause the whole project to fail. While scripts can be implemented by game designers, who typically have little programming experience, advanced AI and soft computing techniques need to be implemented by developers trained in their application.

To address the major drawbacks in soft computing AI implementations, a system must take into account the following:

The steps taken by the system in the decision making process must be observable.

The designers must not lose control on the desired behaviour of the characters.

The ability to manage the degree of adaptability without introducing unpredictable behaviour or excessive learning time must be present.

An appropriate solution is one that can maintain control and observability that is inherent using scripted techniques, yet have the flexibility to learn and adapt based on the characters own in-game experiences. The game developers need to know that they can deliver what is required in a game and not be in a position where the proposed solution for controlling the game characters may or may not work. This is the main reason why scripting has been so prevalent within this realm. It is easy to understand, easy to control and easy to fix when things go wrong.

**Current techniques for controlling character behaviour in computer games**

Interactivity between human players and computer characters has become a large component of modern computer games, and is especially prevalent in Role Playing
Games (RPG). In this genre of game, the player takes the role of a character within the game and interacts with a variety of Non Player Characters (NPC). Some examples of Role Playing Games are The Elder Scrolls 4 (Bethesda Game Studios, 2006) and Baldur's Gate (BioWare, 1998). There are other games where the human player does not interact with characters, as in a puzzle game, or a simulation game. In these cases there are no characters in the game, only the game itself, and the opponent is the computer. There are still more games where there are multiples of any one opponent; we refer to these opponents are Computer Controlled Agents (CCA). These differ from NPCs in that their behaviour is expected to be generic and identical, with little or no dialogue, whereas an NPC usually has unique or prescribed behaviour and a unique set of dialogue options and responses. This thesis focuses on the behaviour of CCAs, but in later chapters we describe how the same application can be applied to NPCs and computer opponents also.

Some common ways of specifying a CCA’s actions are through a script or using state machines (Rabin, 2002). When a CCA behaviour is contained within a script, its actions are defined as a set of condition-action groups. When a certain condition occurs, a specific action or set of actions are performed. When a CCA’s behaviour is controlled by a state machine, a CCA will perform a specific set of actions until a condition occurs. When this happens the CCA will perform a different set of actions, and so on. Both of these methods are in fact quite similar in the approach they take and the resultant outcome. The advantage to using this approach is that the responses are predictable and are easy to manipulate during and after game development. While these characteristics are advantageous to the games developers, they can also be detrimental to game play.
In First Person Shooter (FPS) and Role Playing Games (RPG), where typically a player must battle hundreds or thousands of CCAs, the player often finds one strategy that is most effective and uses this strategy repeatedly. In many games the case is that the scripted behaviour of the CCA's cannot accommodate countermeasures for all strategies that are adopted by players. If the player finds a flaw or weakness in the computer's strategy, then the player has an advantage that they will exploit to try to win the game. This often adds a boost to the entertainment factor of the game, as the human player has found a way to outsmart the computer. However, if this exploit goes unchecked, then the game is not played as the developers had envisioned. Sometimes this flaw or weakness is a result of a script being written incorrectly or a flaw in the game implementation. The computer cannot adapt or compensate for this flaw as it was not designed to do so. This is often fixed by patching future releases of the game.

Most games employ a strategy of increasing difficulty, where the player must compete against a more difficult opponent. This difficulty is almost always predetermined and has no bearing on the strategies or tactics employed by the player during the game. E.g. the attributes of the CCA may be increased, the amount of CCAs may be increased, or the behaviours of the CCAs may change. Other techniques include adding new components to the game, such as weapons or skills, or an experience system where the CCA's attributes are related to the amount of experience the human player's character has accrued.

**The quest for a more engaging opponent**
Traditionally the majority of games are designed to be played by one player against the computer. Along with this is some plot that gives the game a theme, definition and
structure. Most games are still developed with this structure, the most successful games being the ones that have combined an engaging storyline with superior graphics and challenging opponents.

With the advent of multiplayer gaming it has become possible for players to compete against other human players and not just the computer. Games like Quake (id Software, 1996) and StarCraft (Blizzard Entertainment, 1998) brought a new dimension to games where the player was not limited to playing against a computer, but against other human players. This has become a massive revolution over the past few years, as players find that competing against another human player is far superior than the limited and often predictable computer controlled opponents. Games like Counter-Strike (Valve Corporation, 2003) and Halo (Gearbox Software, 2001) have brought game tournaments online where players compete against friends and strangers from around the globe. Leader boards and team competitions are now a common event in multiplayer games.

In recent years Massively Multiplayer Online games (MMO) have been developed where players compete in a persistent world. These games consist of an online environment complete with economies, a variety of plot lines and NPCs and CCAs that inhabit the game. This adds a new dimension to a game which is not only populated by NPCs and CCAs, but also human players. It is this ability to interact, compete and cooperate with other human players that have made these games so successful. Some examples of MMO games are EverQuest (Sony Online Entertainment, 1999) and World of Warcraft (Blizzard Entertainment, 2004).
Aside from storyline and graphics, the most popular approach to creating a more engaging game is by adding more complexity. Whether this is by the amount of tactics available to the computer player, the size and diversity of the scripts used to control NPCs and CCAs, or the customizability of the protagonist in the game. Many games add diversity by adding more types of CCAs into the game. As an example Elder Scrolls 4 (Bethesda Game Studios, 2006) had an immense array of different NPCs and CCAs, many with unique behaviours and attributes.

It may be the case that scripting, while it is an excellent way for defining the behaviour of a CCA, may not scale up to the complexity requirements of modern games. CCAs generally operate from a set of scripted behaviours, one script per type of CCA, so if a player encounters the same type of enemy in the game, the behaviours of these enemies are identical. Most RPG and FPS games will have the vast majority of CCAs have the same behaviour with only a limited number that are scripted uniquely, these being identifiable in game as primary constituents to the plot.

Games such as The Elder Scrolls 4 (Bethesda Game Studios, 2006) and The Sims (Maxis, 2000) had a lot of development go into the behaviours of the NPCs in the game, to such an extent as you can observe their unique behaviours. Examples of such would be going to their respective jobs in the morning, going to sleep at night and eating when they are hungry. While this may appear as intelligent behaviour, these characters are still following their scripted behaviour and will not deviate or adapt unless directed to do so.

**Adaptation of opponents in video games**
Video games over the last decade have seen an enormous change from having straightforward goals and objectives, to interactive characters and immersive storylines.
Encouraging players to continually search for new strategies as the game AI evolves and adapts is something that is not yet realised in the modern video game. The major advantages to this are that the opponent is much more diverse and unpredictable, adapting to all different playing styles. Also the developers do not need to pre-empt all conceivable strategies that the human player might adopt, and usually due to development time constraints, there are only a few different scenarios that are accounted for.

Adaptation of computer opponents in modern computer games is still conspicuously missing from the repertoire of modern enhancements. Some of the reasons for this are:

The perceived amount of effort to implement an adaptable and intuitive AI outweighs the advantages gained by it.

It is easy to fake learning and adaptation. If a player cannot observe the computer opponent adapting, or it can be simulated at a fraction of the cost, then this is often the more attractive route for development companies.

Learning algorithms and other advanced AI techniques often require large allocations of computational resources, something that is a precious resource in video games.

Soft Computing techniques by their nature can be unpredictable and hard to control. The processes involved can be viewed as a "Black Box" and are very difficult, if not impossible, to change or understand why a certain output is the way it is.

There are very few implementations of adaptation in any major video game released to date.
Within the mainstream video game market, there have been some attempts to create games based around Soft Computing techniques. Most employ some kind of learning mechanism, be it neural networks, genetic algorithms and some form of reinforcement learning. In general, online learning and adaptation takes time, time that most games cannot afford to expend. In games like Black and White (Lionhead Studios, 2001), this is mitigated as the basis of the game is the gradual training and learning of the player’s character. Other games use soft computing techniques off line to train the CCA before including the generated behaviour into the game. An example of this is Colin McRae Rally 2.0 (Codemasters, 1998), where the computer controlled rally cars learned to drive around circuits using an Artificial Neural Network (Hannan, 2001). This meant that the game developers did not have to create rules for every individual track. It is entirely plausible that the cars could learn to drive on new circuits using the same techniques, improving the longevity and attractiveness of the game.

It might be enough just to develop a system that is able to switch strategies during game play. The agent does not have to learn, it just has to identify what strategy the opponent is using and select the best counter-strategy. This behaviour is already accomplishable by the scripting techniques described earlier. The only caveat is that the same behaviours will always be applied to counter the strategy used by the player. A more interesting approach is to train the CCAs beforehand with various strategies so that when the need to adapt to the player occurs the agent does not have to learn, only apply what they have already learned. Most scripts are comprised of a set of tactics that come from some library of actions that are available to the specific CCA. (Ponsen, 2005) has shown that CCA can dynamically generate their own scripts in response to in game actions.
It is unrealistic to think that computer opponents can learn everything they need to know through game interactions with human players. Training is very much a part of many soft computing systems. Games like Black and White (Lionhead Studios, 2001) are unique in the fact that the players characters begin as a blank slate and only learn what the player teaches them. The incorporation of learning into computer controlled opponents should be prevalent in video game evolution over the coming years.

**Offline learning and online adaptation**
For an AI to be useable in a game environment, it must be trained to be able to play that game. It is not enough that the AI starts with no prior knowledge and as a result the game is unplayable at the start. No player would want to race against cars in Colin McRae Rally 2.0 (Codemasters, 1998) if the opposition cars were not able to complete a race. In the case of Artificial Neural Networks and Genetic Algorithms, they need to be trained to play the game before it is delivered to the consumer.

Instead of online learning, where the CCA needs to develop previously unknown techniques, a more effective approach would be online adaptation. This would involve the CCA employing pre-learned techniques in response to certain events and game states. The CCA has learned all that it needs to know to operate effectively in the game environment, when presented with certain problems it can adapt what it has already learned, or recombine tactics to create something new. In some cases this may be worse than the current strategy, but in other cases, it might be an improvement. If it finds a certain strategy is effective against a human player it will use this strategy more, forcing the human player to adapt. This way we do not have to control or train the agent's behaviour, just give it the tools to use all the strategies at its disposal. When the agent has
been trained sufficiently offline, once implemented in game, the agent can use what it has learnt to adapt to the current player's strategies. There should be minimal need to "learn" in game as the trained agent should be considered competent enough to perform game actions that are pertinent.

Once the game has begun to be played, it is at that point that the adaptation can begin to fine tune what it has learned off line to the current opponent's game style. This means that out of the box, all players would receive the same experience, however after some game play, the opponents would change their behaviour in response to the actions taken by the player.

**Unpredictability and the illusion of intelligence**

The ultimate goal for any computer game AI is to be indistinguishable between itself and a human opponent. This has a great semblance to Alan Turing and his test of intelligence (Turing, 1950), where messages are passed to and from a room and the resultant responses are interpreted and assessed whether a human or computer constructed them.

There is a fine line between displaying unpredictable behaviour and seemingly irrational behaviour. CCAs are expected by the human player, to be predictable as traditionally computers behave in a logical manner, and unpredictability, cunning and guile are not in their repertoire. If the CCA does something unpredictable or that does not make sense to the player, this behaviour is often attributed to bugs, or poor design. This is one of the major roadblocks to the adoption of learning into video games. The perception is that quality assurance would prove impossible if the in game agents were continually evolving and having unpredictable behaviour.
In modern games, it is often enough for a CCA to seem like it is making rational choices; "The ability of a system to behave unpredictably within the bounds of reasonableness gives a very strong semblance of intelligence." (Barnes & Hutchens, 2002). Having an unpredictable component in a game such as agents that learn and adapt, introduces various challenges for both the testing of the game, and ensuring that the end user is playing what seems like a competent opponent. If learning and adaptation are to be employed by the CCA, it should not be to the detriment of the playability of the game. Some minor unpredictability is tolerated by players, but consistent irrational and random behaviour does not make for a challenging opponent.

**Conclusion**
Learning and adaptation in computer games is a challenge that still remains largely unrealised, from the early attempts at computer AI to the advanced scripting and other character control techniques of modern games. There are arguments as to whether this level of learning is really required to provide a sufficient amount of entertainment as expected by the end user, and the computer game industry certainly prefers the status quo of tried and tested techniques. It does however seem like the next level of evolution of game development is within the realm of advanced AI techniques, with some forays already providing interesting results. For an implementation of learning and adaptation of computer games characters to take off, the reservations of the computer industry need to be addressed, and a level of observability and control need to be part of any tenable solution. With this in mind, the way forward for this next generation of intelligent computer opponents can be realised.
Chapter 2 - Artificial Immune Systems
In this chapter we introduce the concept of the Artificial Immune System (AIS), a biologically inspired construct based on the immune system. We look at some implementations and discuss the different variations and components of the AIS and how it compares to a biological immune system. Finally we describe how an AIS can be used for dynamic strategy generation, and how this can be applied to learning and adaptation in video games.

The Biological Immune System
The main function of an Immune System, is to monitor and identify threats to the host system and, once a threat is detected, generate an appropriate response in the shortest amount of time. In a biological organism, this might mean that an infection is wiped out before it kills the host. The human Immune System has a variety of features which enable it to, very effectively, deal with almost all microbial threats to the body. The most important is probably the ability for the Immune System to adapt to find the most effective response to the current threat without knowing beforehand how to neutralize the threat. Another important feature is that when that same threat happens again, the Immune System retains the knowledge from the previous encounter to be able to fight off the invasion almost immediately.

The fundamental components of the Immune System in vertebrates are white blood cells. These white blood cells can be broken down by various functions that they perform in the Immune System.
B Cells (or B Lymphocytes) are responsible for producing antibodies. Each different type of B Cell produces a different antibody. They are also responsible for identifying threats, with each B Cell having a surface receptor which it uses to recognize antigens.

T Cells (or T Lymphocytes) regulate the immune response by activating other immune cells. They also patrol the host looking for threats and neutralizing them.

Antibodies are produced by B Cells and are responsible for neutralizing their corresponding antigen. The antibodies produced by each B Cell are of the same shape as the surface receptor.

Figure 1 Antigens and how the Antibody recognises them. Source: http://en.wikipedia.org/wiki/Antibody
The Immune Response
When a B Cell encounters an antigen that activates it, an immune response is initiated. The intensity of the response corresponds to how strongly the B Cell and the antigen interact. This is known as the level of affinity between the B Cell and the antigen. The higher the affinity, the stronger the immune response.

After a B Cell is activated it begins to clone itself in a process known as clonal selection. The clones undergo a process called somatic hypermutation, where the clones will mutate to a varying degree based on the level of affinity. The process of clonal selection followed by somatic hypermutation is collectively known as affinity maturation. (Dasgupta & Nino, 2008) It is so called as the process is governed by the affinity between the cell and the antigen. The greater the affinity, the stronger the response, which includes a greater number of clones with a smaller rate of mutation. Conversely, a lower affinity will result in a lower number of clones and a higher rate of mutation. (Berek & Ziegner, 1993) Some of these clones become memory cells which remain in the body long after the threat is eliminated. The memory cells enable the Immune System to respond quickly to future attacks by elements that produce the same antigens.

The Artificial Immune System (AIS)
Inspired by the human immune system, Artificial Immune Systems are a computational construct that compromise of the main elements of the immunological processes. The most significant of these being self non-self discrimination, or the ability to differentiate between elements of the system and those that are a threat to the system. It is this concept that is prevalent throughout All Artificial Immune implementations. Along with this is the concept of affinity, or the how appropriate the interaction between the antibody and antigen, learning and memory. Wikipedia describes the field of AIS as:
"The field of Artificial Immune Systems (AIS) is concerned with abstracting the structure and function of the immune system to computational systems, and investigating the application of these systems towards solving computational problems from mathematics, engineering, and information technology. AIS is a sub-field of Biologically-inspired computing, and Natural computation, with interests in Machine Learning and belonging to the broader field of Artificial Intelligence." (Wikipedia - Artificial Immune System)

There are various different ways that the elements of the immune system have been represented. There is the B and T Cell models that model the processes of the B and T Cells of the immune system (Castro & Von Zuben, 2000), Immune Network Theory based approaches (Jerne, 1974) (Perelson, 1989), and Danger Theory (Matzinger, 1994).

Immune network theory was proposed by Jerne (Jerne, 1974) as a way to represent the ability of the biological immune system to identify, respond to and remember threats to a system. Artificial Immune Networks are based on this theory and represent a network of antibodies which the affinity of the antibody is used to weight the connections within the network. Artificial Immune Networks have been used in tasks such as data mining, function optimization and image recognition. An example of an Artificial Immune Network implementation is aiNet (de Castro & Von Zuben, 2003). aiNet generates a network of antibodies to which an antigenic pattern was presented. The antibodies which generated the highest affinity were selected would undergo the processes of clonal selection and mutation. Following this, the antibodies with the lowest affinity were culled from the network. After this process a number of new randomly generated antibodies would be added to the network (metadynamics). aiNet is largely used for data analysis and compression (Stibor, 2007)
The danger theory model does not take the form of self-non-self discrimination like the B or T cell models, instead it is a reactive system that identifies "danger" or threat to the system. (Matzinger, 1994) theorized that the danger theory model would provide a flexible and localized way that an immune system could respond and adapt to the changing self instead of being an army always on patrol for any foreign invaders.

Aickelin and Cayzer (Aickelin & Cayzer, 2008) have described Danger Theory as a response that does not need to treat all foreign entities or immune activators (antigens) as threats to the system. Instead the immune response is localized to an area where danger has been detected. An example of which is a damaged cell, either by a foreign source or self initiated will be identified as being a danger, and hence the target of an immune response. They note that the danger theory model is less about how the immune system identifies threats to the system, but which threats are important and should be dealt with. Feature selection is not relevant, only the signal that there is something bad (or something good, in applications such as data mining) being received and dealt with appropriately.

**Clustering:** Using Artificial Immune Systems for clustering and data analysis is an interesting concept as there does not seem to be a correlation between the biological immune system's functions and data analysis. The aiNET Artificial Immune Network developed by Castro and Von Zeuben (Castro & Von Zuben, 2000) is construct that uses an Artificial Immune Network for large scale data compression and analysis. Tang and Vemuri (Tang & Vemuri, 2005) compared document clustering performance of aiNet with other clustering techniques such as K-Means and HAC (Hierarchical Agglomerative Clustering). They described how documents could be represented by a vector matrix of features, which then was presented to the aiNet algorithm as an antigen. Their results
showed that aiNet gave better or at least comparable results to HAC or K-Means. The best results were obtained when the document feature set was more varied than when the documents were relatively similar. They concluded that the best application for aiNet in document clustering was when there were large size document sets which contained a large amount of redundancy and noise. Li, Gao and Tang (Li, Gao, & Tang, 2009) showed that even though Artificial Immune Networks were good when dealing with noisy data, the performance could be improved even further if fuzzy logic could be applied when identifying the antigen inputs.

**Data Mining:** Data mining is another realm to which applications of Artificial Immune Systems have be applied. one of the initial implementations of Meng et al (Meng, van der Putten, & Wang, 2009) implemented a series of comprehensive benchmarks to determine how comparable an Artificial Immune System based document recognition system is to other forms of data mining. The Artificial Immune Recognition System (AIRS) was originally described by Watkins (Watkins, 2001) and is a cluster based classifier that performs well through finding the optimal number and position of cluster centers. The process that was used to classify the data was that initially the AIRS was trained with a sample from the data set. This sample was compared with a memory cell and based on the affinity the memory cell would clone and mutate. After the training was complete, the data was classified based on the response of the data with the memory cells. A high response meant that the classification was the same or similar, whereas a low response meant that the data sample was in a different class. Finally the most stimulated memory cells were classified using the K Nearest Neighbour method. The results show that in
comparison to other data mining techniques that the Artificial Immune Recognition System performed comparably.

**Network intrusion detection:** There is a strong analogy between intrusion of a biological system by a foreign entity and a computer network being attacked by external elements such as hackers or viruses. Hofmeyr and Forrest (Hofmeyr & Forrest, 1999) used an Artificial Immune System to act as an Intrusion Detection System (IDS) for a computer network. Within the IDS, network requests are encoded into a binary string and then inspected to determine if the request shows anomalous identifiers. Positive identifications were stored as memory cells and used in later identifications. These memory cells would have a lifespan of about 3 months, after which it was determined that the threat was not recurring. Dal et al. (Dal, Abraham, & Abraham, 2008) expanded on this by combining the AIS learning with a genetic algorithm to evolve the detected responses. They dubbed their system IDS-EVOLUSIRS in reference to the evolution of a secondary immune response, and contained for major stages:

- Encoding of network data into a comparable binary string.
- The training and generation of anomaly detectors
- The detection of intrusions to the computer network
- The creation of memory cells which remembered past anomalous network actions

Both methods had positive results in that after the first interaction with an anomaly that the secondary immune response was improved enabling faster and more accurate detection of network intrusions.
**Spam filtering**: The separation of spam from genuine email is one area which remains a constant problem in modern e-society. There are numerous approaches to combat this problem, most being quite successful, however a completely accurate solution has not been found yet, with most applications using a combination of methods to filter the spam. There have been a few approaches to spam identification proposed that use Artificial Immune Systems. The AIS seems quite appropriate for this type of duty as it is especially adept at identifying self and non-self elements (i.e. non-spam emails versus spam emails) and adapting to constantly modify its ability to recognize foreign threats to the system. Yue at al. (Yue, Abraham, Chi, Hao, & Mo, 2007) describe a system where they use an aiNet (Castro & Von Zuben, 2000) implementation to cluster and filter out spam from an email data stream. They present the email data stream, representing a collection of antigens, to their Artificial Immune network (ICAINet) which then classifies and clusters the emails based on the probability of it being spam. To allow the Artificial Immune Network to process and learn the emails, the stream is divided into time separated windows. The results of each window are then incrementally added to the clusters (hence the I and C in ICAINet) that were generated from previous windows. Their results showed that they were able to cluster the emails better than a standard clustering algorithm using their incremental approach. Also they were able to achieve better quality cluster when the homogeneity of the emails was decreased and or the distance between clusters increased.

Oda and White (Oda & White, 2005) describe a system where the they use regular expressions as antibodies to classify email as spam. As the AIS system learns and adapts to identify more and more spam, so too do the regular expressions evolve. They term
their antibody and the count of spam versus email detected as a "digital lymphocyte". These lymphocytes are trained as the system processes incremental amounts of emails. The emails processed were from the SpamAssassin corpus (SpamAssassin, 2005), and they found that their system was able to classify spam at about the same rate of other contemporary systems. What was impressive about their system was that it was only the AIS that was performing the classification, whereas in most other systems, multiple methods are combined to identify spam, making this an ideal solution where resources or processing power is limited.

Robotic control and learning: There has also been some research into embedding Artificial Immune Systems into robots, wither to achieve autonomy or to implement self diagnosis and self healing attributes. Singh and Nair (Singh & Nair, 2005) implemented an adaptive learning mechanism, incorporated into a pair of robots, that when one robot entered detected a danger or warning signal, a second robot would be able to react and assist. The "Learner" robot was then able to modify its behaviour (movement speed in the experiment), and adapt so that when similar dangers were presented in the future, the robot's speed could be reduced accordingly. This was to exhibit that the robot control and learning was analogous to the innate immune systems in biological entities.

Neal et al. (Neal, Feyereisl, Rascunà, & Wang, 2006) also implemented a system in which robots were able to detect and self diagnose issues within the system. They expanded on the notion of "inflammation" where the rate of the increase of danger to the system would be responded to with an equivalent measure of response to that danger. An example of this could be that the robotic system detects a sudden rise in temperature in a motor and the response would be to increase the cooling fan speed in proportion to the
rate of rise in temperature. Or if extreme overheating occurs to turn off the motor entirely for a period of time to prevent critical failure. Their investigation focused on the properties of the innate immune system and showed that a combination of TLR (Toll Like Receptors), inflammation and localization, was able to efficiently and effectively provide a comprehensive level of danger response.

Kim et al (Kim, Greensmith, Twycross, & Aickelin, 2010) used the danger theory model to identify malicious code execution within a system. The danger was established when certain events were generated within the system such as:

- Violation of the system security policy
- Fluctuations in memory and/or CPU usage
- An unexpected increase in the current workload of the system.

They used the AIS in conjunction with the current state of the system to determine whether system calls were malicious. These "attack signatures" once identified were encoded in a policy and added to the list of the system security policies. In this way the system itself could identify and neutralize threats while protecting from future attacks. Unlike a virus detection mechanism, which identifies and neutralises threats to the system, but cannot learn to protect itself from infections of the same virus, the AIS implementation is able to provide an "immunity" from future identical or similar malicious attacks.
Dasgupta and Nino describe in their book some applications of AISs in real world scenarios such as virus detection, intrusion detection systems, fraud detection, robotics and control and anomaly detection to name a few. (Dasgupta & Nino, 2008).

**Components of the B Cell model**

There are four main components of the B Cell Model of an AIS; recognition, response, adaptation and memory, each of which are necessary for most immune system implementations.

**Recognition:** A threat can be classified as anything that intends to upset the natural operation or homeostasis of a system, biological or otherwise. This could be anything from a virus attacking an organism to a computer hacker trying to break into a secure environment. If there are no active countermeasures to the threat, the results can be catastrophic. There are several ways to identify and counter threats. With respect to AIS, the domain of the threat is all possible negative interactions with the system. The domain of the AIS is all possible counteractions that can be generated from a pre-existing library of actions. If a threat exists that no combination of counteractions will contain, then the threat continues. We can see this in the human immune system, when our body encounters pathogens for the first time, and so has little or no defence against. However after the human immune response kicks in and hopefully destroys the pathogen, then future encounters are much less severe and often not even noticed. Our Immune System has the potential to combat any threat where a counteraction can be constructed from within the library of genes that act as the building blocks for antibodies.

**Response:** Once a threat is detected the appropriate response, if any, needs to be initiated. The appropriate response is the one which neutralises the threat in the quickest and most
efficient manner. This way the least amount of harm is incurred by the system. Partial responses can still be effective, by inhibiting the attack and allowing the Immune System time to find a more appropriate response. More damage may occur to the system and recuperation time will be longer. How an AIS evaluates the response is through finding the appropriate counteraction for the threat. Many different combinations of genes are tried, the ones that are the most successful then cause identical clones to be produced, whereas less effective counteractions are inhibited and therefore free up resources for the selected counteraction. Biological systems usually have a limited amount of resources to devote to immune responses, so the current most effective response will be the one that consumes most of the resources dedicated to the overall immune response.

**Adaptation:** There is a certain amount of variability in the immune response. As the immune response progresses, the most effective components of the response are focused on as these are most likely to lead to a swift resolution to the current attack. This response is a gradual progression of attaining a solution that will wipe out the invading pathogen. Adaptation is done through the mutation of the cells which produce the most successful antibodies. The more successful the immune cells are, the less they mutate. A response that is 100% successful will not need to mutate at all as the best possible response has been found, i.e. a higher affinity will cause the solution to mutate less, and will generate clones that are quite similar.

**Memory:** Once a counteraction has been successful in neutralising a threat, then in the future, if that threat ever occurs again, then the response is much faster and more effective as the AIS does not have to perform the same search and test process. Instead
the counteractions that have been successful in the past are much more likely to become part of the mainstream counteractions that are ever present.

Timmis and de Castro (Timmis & de Castro, 2003), define a framework for computationally inspired models. This framework consists of the following:

- A representation for the components of the system.
- A set of mechanisms to evaluate the interaction of individuals with the environment and each other.
- Procedures of adaptation that govern the dynamics of the system.

Along with this there has to be some way to represent the antibody makeup that determines how the elements in the system can be compared to each other. An example could be applying a boolean operator two binary strings, or as in Table 1, two strings generated from the alphabet.

<table>
<thead>
<tr>
<th>Antigen</th>
<th>A</th>
<th>H</th>
<th>T</th>
<th>G</th>
<th>Y</th>
<th>U</th>
<th>E</th>
<th>C</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antibody</td>
<td>B</td>
<td>E</td>
<td>T</td>
<td>H</td>
<td>Y</td>
<td>U</td>
<td>E</td>
<td>B</td>
<td>N</td>
</tr>
<tr>
<td>Affinity</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1 Comparing two strings from a library composed of {A ... Z} where the same character in the same position increases the affinity by 1, for a total of 4 in this case.

In the next chapter we will show how these components can be used together to create a generic model for an AIS.
There are various different approaches to modelling an Immune System, with each approach trying to capture some property or characteristic of the biological immune system. Some focus on response and adaptation, whereas others model the process by which the immune system communicates a threat throughout the system, or how the immune system recognises self cells compared to foreign or harmful cells. An AIS implementation may be one or a combination of the following: (Garrett, 2005)
Clonal Selection: This model describes how random elements are created and when one of these encounters a threat, then they clone themselves to combat the threat. The ones that are not activated become less likely to be produced.

Negative Selection: Immune elements that are reactive to self (the body in which they reside) are rendered inactive. However these cells may also be needed to combat a threat in the future.

Danger Models: This theory suggests that cells undergoing unnatural cell death send out a danger signal initiating an immune response. The danger signals are localised and the intensity of the signal can determine the response.

Immune Network Models: This is a model of how the Immune System inhibits or excites responses based on threats. The connections between the antigens is often represented by the level of affinity.

All these approaches have the benefits and disadvantages, but the one that seems most appropriate for adaptation in computer games would be a combination of Clonal Selection and Negative Selection.

**Artificial Immune System for controlling video game opponents**

An Artificial Immune Systems can be used to generate adaptive responses to a human player's strategy in a computer game. Some scenarios where this would seem appropriate are games where the human player is attacking the computer player to gain some objective. Also games where resources are involved, like in RTS games, the resources are used to build defensive and offensive units to defeat the computer opponent. An AIS can
be used to respond to the human players activities and to learn and adapt to find the right 
solution that will cause the human player to ultimately lose the game.

This benefits the game by not having a predictable opponent and adapting to strategies 
that will continually defeat the computer. The goal of this section is to generate a 
collection of strategies (B Cell configurations) that will cover a wide range of responses 
to a players actions. If a strategy is implemented by the human player where an 
appropriate response does not exist then the system should be capable of adapting and 
generating a strategy that is at least more effective than the existing ones. The human 
player may have a strategy that they employ to defeat the computer opponent. The 
affinity of this strategy to the computer's strategy can be thought of in terms of how 
effective the strategies are against each other, e.g. if the computer always loses then the 
affinity is extremely low. If the computer always wins then the affinity is high, and this 
will lead to the human player being required to change strategies. As in the biological 
immune system, the winning computer strategies can be remembered and stored for 
future use so they do not have to be relearned.

A way to achieve this is to have a collection of actions that are available to the computer 
player, then when presented with certain in game scenarios it is able to select the 
appropriate response from those actions. This takes the responsibility from the developer 
to generate complex scripts that control the agent's behaviour.

The format for training the computer agents is as follows:
The agents are initiated with a selection of tactics available to them
They fight against each other and develop a set of strategies and counter strategies using AIS

Once the strategies are sufficiently learned the agent is deemed as fully trained and available for use in the real game.

As in the biological immune system, an AIS only has a limited amount of resources available for it to use in its response, whether this be limited by the hardware it is running on, or by restrictions imposed by the game dynamics. One thing that make the AIS a good choice for using is computer games, is that it can find a good, if not optimal solution in a relatively short period of time, and maybe this is all that is required to give the human player a challenging yet enjoyable gaming experience.

**Starting with a pre-populated current selection of successful strategies**

When we address the issue of speed, or lack thereof, that AI learning or adaptation brings to computer games, a large proportion of computational resources are taken up through the learning process. This usually involves some form of searching of a domain space to find a solution that matches or exceeds some threshold. The AIS can mitigate this somewhat as the AIS can have a priori strategies or responses that have been pre learned offline. Depending on the parameters of the AIS, the more strategies it learns the more general its response will be. So in a game where the human opponent has a very large number of strategies available to them, this would be the best approach as the AIS is more likely to have previously encountered a strategy that the human player will choose. In a more restricted game, it may only be necessary for the AIS to learn a few strategies, this way it can adapt and respond very quickly to the human opponent. Training offline in
this way has the advantage of giving any learning that the AIS must do an advantage over having to perform all learning in-game.

**Conclusion**
Artificial Immune systems and their biological counterparts are excellent tools for performing a quick and effective response to any threats it has been designed to repel.

The combination of learning and adaptation are combined in such a way so that available resource are devoted to where they are needed the most. The scope of the AIS is only limited by the library of actions available to it and any resource limitations that may be imposed by the system in which it resides. This should make an AIS an ideal solution for learning and adaptation in a real time computer game environment.
Chapter 3 - Artificial Immune Systems for strategy generation

Chapter 3 covers the structure and elements that make up an Artificial Immune System, and how this compares to the biological immune system. We look at the process that the AIS undergoes when performing adaptation, and how our implementation produces a collection of strategies each generation. The environment that the AIS implementation is discussed and how different game environments can be supported by augmenting sections of the AIS. Finally, we introduce our testing platform and talk about the testing process that is used to evaluate the performance and level of adaptation of the AIS in a game environment.

The structure of the AIS

The Artificial Immune System that we can use for strategy generation is based on the B Cell model. This model of a biological immune system comprises of the following components (Dasgupta & Nino, 2008):

- Maintenance of a specific memory set
- Selection and cloning of most simulated antibodies
- Removal of poorly stimulated or non-stimulated antibodies
- Affinity maturation (hypermutation) of activated immune cells
- Generation and maintenance of a diverse set of antibodies

All of the above are required in our AIS implementation to adapt to the opposing strategy and to remember previous strategies that were successful.
The biological immune system works by constantly generating B Cells which circulate throughout the system awaiting activation by some antigen. Once activated these cells go into an active state where they clone themselves many times. The clones try to adapt and become a better match to the antigen by undergoing slight mutations, which may or may not be a closer match with the antigen. This means that the Immune System targets resources only to current invasions, while inactive cells remain circulating until they themselves are activated by some other antigen.

The following test simulate in a simple way the production of B Cells from the library of genes. Some immune cells work better than others and the cells that form the best strategy for recognising the antigen are more apt to be generated. The human immune system contains a vast library of genes from which B Cells can be programmed. Combinations of these genes is what gives our immune system the potential to counter almost any microbial attack on our body. In an AIS, the diversity of the strategies generated are dependent on the scale of the library of tactics available to it.

**Memory:** The AIS maintains a collection of strategies and their affinities through the course of the adaptation process. As the AIS adapts to opposing strategies, so too does the collection of successful strategies change.

**Clonal Selection:** If a strategy has a low affinity then the probability that tactic will be selected in future generations is lowered, whereas if the affinity is high then it is more likely that strategy will be selected to populate the current strategy selection.

**Meta Dynamics:** This is the process of evaluating the current strategy set at the end of the generation. The worst performing strategies will be removed from the successful
strategy collection to make room for strategies that are more successful against the current opposing strategy.

**Mutation:** When a successful strategy has been selected to populate the list of current strategies, that strategy undergoes mutation. This mutation happens inversely proportional to the affinity of the AIS strategy to the opposing strategy.

To make our AIS implementation independent of any particular platform, it has been separated into components that can be abstracted out to provide an interface to different games, for example.

![Diagram of AIS Implementation](image)

**Figure 2 The components of the AIS showing within what realm each of the components lie**

Figure 2 shows where each of the individual components of the AIS reside. The Clonal Selection, Affinity Maturation and Meta Dynamics components are all contained within the AIS processor, and do not change based on the environment the AIS is implemented within. The Successful Strategy collection and AIS Parameters are unique to each AIS Implementation and are determined by the implementer of the AIS system. The Mutation Implementation, Evaluation Implementation and Tactic Database are all environment specific. The Tactic Database is determined by what environment the AIs in implemented...
within, i.e. what moves are allowed within a game. The Mutation and Evaluation Implementation are also dependent on the environment that they are implemented within.

**AIS initial implementation**
For Artificial Immune Systems there are a number of parameters that affect the behaviour of the AIS, which will impact the performance and results that are generated. In our implementation environment the parameters are:

**Generations:** The number of times the population is presented to the opposing strategy

**Population Size:** The amount of individuals that make up the current selection of strategies.

**Tactic Library:** The tactics that have been selected by the game developer to be combined into a strategy. This can be as simple as a single command (turn left), or can be a full independent strategy in itself.

**Maximum Strategy Length:** The strategy length represents the maximum amount of tactics that are allowed into a single strategy. For basic tactics, this number should be high to allow for more engaging strategies. For more complex tactics, this number does not need to be high and can be as low as one. The amount of tactics can vary from one strategy to the next, but cannot exceed this value.

**Mutation Rate:** The mutation rate is a value between 0 and 1 that determines how much or how little mutation occurs to a strategy during the clonal selection phase.
Maturation Factor: The maturation factor is a value between 0 and 1 that determines the rate of change of the same strategy from one generation to the next. The concept for this control is based on Eisenstein's "Scaled Fitness" concept (Eisenstein, 2003).

Structure of the AIS algorithm
We define the process that the AIS undergoes as it develops a set of strategies that are competitive against the current opponent strategy. Each generation will perform one full cycle of the AIS process, from selection to evaluation to removal. Figure 3 shows the full cycle of the adaptation process that the AIS undergoes during each generation of an encounter with an opposing strategy.

![Diagram of AIS process](image)

**Figure 3** The AIS process for the generation of a collection of the current strategies that have been selected to compete against the current opponent strategy

The current strategy collection consists of strategies that have been selected from the successful strategies collection, mutated strategies from the successful strategies collection and randomly generated strategies.
**Update Current Strategies:** At the start of each generation if there are no previous successful strategies then randomly generated strategies from the Tactic Pool will be created up to the population size. Typically this is the case for the first generation if no prior learning has been performed. If there are successful strategies from a previous generation they will be used as the current strategies. If the current strategy count is less than the population size, random strategies will be added until the population size is reached.

**Clonal Selection:** Select the current best strategies from the successful strategy collection. The current selection size must be equal to the population size.

**Affinity Maturation:** An evaluation is done on each strategy in the current strategy collection. This evaluation involves determining the performance of the selected strategy against the opposing strategy. Also during the affinity maturation, each strategy generates clones of itself in proportion to the strategy's current affinity. Also, these clones are mutated in proportion to the affinity and the current mutation rate.

**Meta Dynamics:** During this phase, the strategies who performed poorly against the opponent strategy are removed from the current strategy collection. One way of implementing this is to sort the current strategy collection by affinity and remove the lowest 10% from the collection.

**Return Current Strategies:** The current set of strategies are returned as the result from the AIS process to the game engine as a selection of strategies that are the current best performing strategies as evaluated by the AIS.
Robocode Project
The initial implementation will be performed using the Robocode (Robocode Project, 2001) framework. Robocode consists of a two dimensional arena which has a fixed height and width. Into this arena can be added one or more robots, who are going to battle. The battle lasts a set number of rounds with the results displayed after the final round. A round consists of a set amount of turns, each turn provides each robot with an opportunity to take some action. Some common actions are move forward, turn left, fire, turn gun right etc. Robots locate each other with a radar, and attack by firing a bullet or ramming. Firing a bullet also takes a certain amount of energy from the attacking robot.

If there is only one robot left at any point in time, then that robot is deemed to have won the round. If there are two or more robots left when the round ends, then all surviving robots are deemed to have won. At the end of the battle, the robot with the highest number of wins is declared the winner. At the end of the battle each robot is allocated a score based on its performance.

The reason why this is an excellent environment for the initial implementation is that:

There are already a set of strategies included in the basic game, and a wide range of examples available online

It is designed to be extensible and modification of robot behaviour is fully supported.

The platform has been in existence for ten years with constant updates and revisions, so it can be considered a mature platform.

The engine has been designed to operate in a testing mode where robots can be tested over thousands of iterations in a short period of time.
**Evolution within the Robocode Environment**

There have been attempts to evolve competitive robots within the Robocode environment using soft computing techniques and evolutionary algorithms, in some cases being competitive with human designed robots. (Hong & Cho, 2004) (Shichel, Ziserman, & Sipper, 2005) (Eisenstein, 2003). The goal of this work is to show that using an AIS, that adaptation and learning are possible in an environment that has already shown to be capable of this using other methods such as Neural Networks and Genetic Algorithms.

For these experiments, ten tactics that the AIS Robot can use to generate strategies have been chosen. We identify these tactics by the letters A to J. The tactic represents the basic building block of a strategy and is defined within the scope of the game environment. In this implementation, each tactic represents a fully functional set of actions that can control a robot so that in the event that a single tactic is chosen, the robot will still perform sufficient actions to provide a competitive opponent. Some other reasons as to why this level of granularity was chosen for our AIS system are discussed at a later stage.

The elements that are contained within a tactic are all valid actions within the Robocode Environment. Tactic "A" contains the following actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>ahead</td>
<td>Random</td>
</tr>
<tr>
<td>turnRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>turnGunRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>fire</td>
<td>3</td>
</tr>
<tr>
<td>turnRadarLeft</td>
<td>Until enemy scanned</td>
</tr>
</tbody>
</table>

*Table 3 Actions for tactic A*

A strategy for the AIS robot is one or more combined tactics that will result in a contiguous chain of actions (The list of all Robocode actions can be seen in Appendix B). For example in the strategy ACFG, tactic A directs the robot to move forward a random amount, whereas tactic F directs the robot to move forward 100 units. As tactic F is processed after A, this results in the robot moving forward 100 units. If the same
tactic appears multiple times within the same strategy its actions are applied in the same order, that is to say that the strategy AAA is exactly the same as strategy AA or strategy A.
The strategy ABA would produce the following actions:

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Action</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ahead</td>
<td>Random</td>
</tr>
<tr>
<td>A</td>
<td>turnRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>A</td>
<td>turnGunRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>A</td>
<td>fire</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>turnRadarLeft</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>B</td>
<td>ahead</td>
<td>200</td>
</tr>
<tr>
<td>B</td>
<td>turnRight</td>
<td>90 degrees</td>
</tr>
<tr>
<td>B</td>
<td>fire</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>turnGunLeft</td>
<td>90 degrees</td>
</tr>
<tr>
<td>B</td>
<td>fire</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>back</td>
<td>200</td>
</tr>
<tr>
<td>B</td>
<td>onHitWall</td>
<td>turnLeft 90 degrees</td>
</tr>
<tr>
<td>A</td>
<td>ahead</td>
<td>Random</td>
</tr>
<tr>
<td>A</td>
<td>turnRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>A</td>
<td>turnGunRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>A</td>
<td>fire</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>turnRadarLeft</td>
<td>Until enemy scanned</td>
</tr>
</tbody>
</table>

Table 4 Actions that will be combined to produce strategy ABA

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Action</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABA</td>
<td>ahead</td>
<td>Random</td>
</tr>
<tr>
<td>ABA</td>
<td>turnRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>ABA</td>
<td>turnGunRight</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>ABA</td>
<td>turnGunLeft</td>
<td>90 degrees</td>
</tr>
<tr>
<td>ABA</td>
<td>fire</td>
<td>3</td>
</tr>
<tr>
<td>ABA</td>
<td>turnRadarLeft</td>
<td>Until enemy scanned</td>
</tr>
<tr>
<td>ABA</td>
<td>back</td>
<td>200</td>
</tr>
<tr>
<td>ABA</td>
<td>onHitWall</td>
<td>turnLeft 90 degrees</td>
</tr>
</tbody>
</table>

Table 5 Final actions that make up strategy ABA

Some examples of different strategies are ABGJ, BHIAEF, CD, JEDCAI. Each of these strategies have a different behaviour based on the tactics that comprise that strategy, and the ordering of the tactics. A similar approach to creating strategies by programmatically selecting from a library of actions is taken by Spronck et al. (Spronck, Sprinkhuizen-Kuyper, & Postma, 2003). They were successful in generating random scripts from a rule base that controlled in game characters in their simulated game environment. These
scripts were dynamically adapted based on a fitness measure determined from the outcome of the simulated encounters.

**The Testing Procedure**

For our tests the AIS controlled robot will encounter the three selected opponents over a set number of generations. The criteria that are used to evaluate whether this is the case are:

The AIS Robot can generate a viable or competitive strategy against a static or single opponent strategy.

The AIS Robot can adapt its strategies in a reasonable time frame when the opponents strategy changes.

The robots that were chosen to simulate the human opponent were the top three performing robots from the RoboRumble "Micro" League (RoboRumble Rankings, 2011). They are jk.micro.Toorkild 0.2.4b, kc.micro.Thorn 1.252 and kc.micro.WaveShark 0.31 (which we will refer to as Toorkild, Thorn and Waveshark).

The tests will be run over 10 generations. A generation consists of the collection of selected strategies each being evaluated against the current opposing strategy. Each round is one complete fight to the death between two robots within the Robocode environment. The result is used to calculate the affinity of the AIS generated strategy against the opponent strategy.

Adaptation against a single strategy will involve the AIS robot starting out with randomly generated strategies, and over time these strategies will evolve into a selection of strategies that perform competitively against the opponent strategy. Adaptation against
changing strategies will involve the AIS robot adapting to a single strategy first, then revising its strategies and adapting to a new strategy when the opponent strategy changes. Finally, the opponent strategy will switch back to the original strategy. In this case as the AIS has already encountered this strategy previously, the rate of adaptation should be quicker.

**Evaluation of AIS strategies**
When a battle ends the degree of the success of the current strategy is calculated using the following formula, $r$ being the resultant score which is a signed integer:

$$r = \frac{r^c}{r^c + r^o}$$

where $r^c$ is the result of the current strategy for the AIS robot, and $r^o$ is the result of the opponent robot strategy. The affinity, $a$, is the current level of how competitive the current strategy is against the opponent strategy and is calculated as follows.

$$a_k(t + 1) = a_k(t) + \delta(r_k(t) - a_k(t - 1))$$

where $a_k$ is the affinity of strategy at index $k$, $t$ is the generation at time $t$, $\delta$ is the maturation factor which lies between 0 and 1, and $r_k$ is the result of the encounter of the strategy at index $k$ against the opponent strategy. $k$ is the index of strategy within the current set of selected strategies. The affinity then changes over time with respect to the results that the strategy as attained against the opponent strategy.

The average affinity, $\bar{a}$, is the average of all the affinities generated in one generation. This gives an overall score to how well the AIS performed in that generation. We calculate the average affinity as follows
Typically in Evolutionary Computation, as the system evolves, the results that are generated are more appropriate to the solution. Similarly in our system, as the AIS evolves, the average affinity increases as the system finds better and better counter strategies to the opponent strategy.

**Mutation of AIS strategies**

When a successful strategy has been selected to populate the list of current strategies, that strategy undergoes mutation. The chance that a mutation will occur are proportional to the current level of affinity of that strategy, the affinity being in the range of 0 to 1. If the affinity is high (i.e. greater than 0.75) then there is a lower chance that the strategy will mutate. The reason for this is that this strategy has already proved to be a very good adversary to the opponent strategy, so the need for mutation for this strategy is minimal. If the affinity is low (i.e. less than 0.25) then the chance that mutation will occur is much higher. A low affinity means that the strategy has not been successful. In this case it is probable that mutation will produce a better result than with a strategy that already has a high affinity.

The mutation functions that are used in the AIS Robot are:

Replace a single tactic in the strategy with another tactic that is different from the chosen tactic to replace and does not already exist in the strategy. E.g. strategy ABCDE may become ABCDF.

Swap two tactics in a strategy. E.g. the strategy ABCDE may become EBCDA. The position of the strategy to switch and the destination are chosen at random.

\[
\bar{a} = \frac{\sum_{k=1}^{n} a_k}{n}
\]
Conclusion
An immune system must, at the very least, provide a means of representation of an antibody, a way of evaluating it against an antigen, a means of improving its performance and a way to remember antigens that it has previously encountered. Chapter 3 has described a means of replicating this in the form of an AIS and set out the criteria for what it must do to show that the implementation is accurate. Our implementation has a representation of an antibody, in the form of a strategy, a collection of previously successful strategies, which form it's memory, a mutation and maturation function which will augment the current strategy to try to find a better counter-strategy and finally an environment in which we can evaluate the performance between two strategies and calculate an affinity which controls the adaptation process.
Chapter 4 - Testing AIS adaptation in a game environment

There are various factors and parameters that can change the performance and behaviour of an AIS when learning and adapting to opponent strategies. Chapter 4 investigates how changes in these parameters can affect the rate of learning and adaptation in our AIS implementation.

Evaluation of changes in parameters of the AIS

The following tests will determine what effect, if any, changes in the various parameters of the AIS implementation will have on the performance and behaviour of computer characters. A goal of this thesis is to show that during game development, that the behaviour of characters controlled by an AIS can be augmented by changing the appropriate parameters. These parameters can be set during the development period of the game or changed in-game if required.

The tests will evaluate the adaptation performance in terms of the population size, the maximum strategy length, the maturation factor and the mutation rate. The AIS performance will be evaluated on the level of increase of the average affinity the AIS attains over a set number of generations. We use the term generation to denote a specific iteration in the adaptation process. The performance of a strategy is deemed to have been successful if it's affinity has increased from one generation to the next, and unsuccessful if it has decreased. An average affinity of 0.5 or greater means that the AIS has won more that it has lost, attaining a score of 50% or more for the strategies in that generation.

The AIS parameters are:

**Population Size:** This is the maximum amount of strategies that are evaluated in each generation. At the beginning of a generation, if the current strategy count is less that the
population size, random strategies are added to the population until the population size is reached. The randomization process involves picking from the pool of tactics whose count is randomized between one and the Maximum Strategy Length.

**Maturation Factor:** The maturation factor controls the rate of change of strategies from one generation to the next. The higher the maturation factor, the more likely the result from the current generation will have an effect on the same strategy's result from previous generations. A maturation factor of 1 would equate to having no maturation factor, conversely a maturation factor of 0 would mean that the strategies do not change at all from one generation to the next.

**Mutation Rate:** The mutation rate is the value which determines the amount of change in a strategy. For this implementation the mutation rate is in the range of 0 to 1, higher meaning more mutation, and lower resulting in less.

**Maximum Strategy Length:** The strategy length is the maximum amount of tactics that are present in any strategy. The strategy length can be of length 1…n, n being equal to the Maximum Strategy Length. The higher the strategy length the more diverse the behaviour of the robot will be. For our implementation, there is no restriction on the amount of similar tactics that can make up a strategy.

Global parameters of the Generation Count and the Tactics Library are set for the test environment. In an implementation of an AIS in a game, these parameters would be specific to that game.

**Tactic Library:** These are the tactics that the AIS can chose from to build its strategy. This library is comparable to a gene library in a biological immune system.
**Generation Count:** This is the amount of times that a player will play against a set of strategies developed by the AIS. Typically each generation will produce more appropriate strategies to the opposing strategy, making the game increasingly difficult. We keep this low for our tests as we want to be sure that the AIs will adapt in a observable amount of time. For these test a generation count of 10 was chosen. We decided that this is what might be an appropriate amount of engagements to adapt to a players strategy. To the human player it would not be evident that the computer opponent was adapting to their strategy if the computer opponent's strategy did not change after a certain number of encounters.

As described in Chapter 3, these tests are performed against each of the top three contenders in the Roborumble, Microbot league (RoboRumble), Toorkild, Thorn and Waveshark. The average affinities of all three encounters are then used to determine how the AIS strategy has performed.

**Change in Population Size**
The population, as described here, is the amount of strategies that are evaluated in each generation. Each member of the population has a chance of being mutated in proportion to its affinity and mutation rate. A larger population size gives rise for more diversity when performing affinity maturation, as there will be a higher number of mutated strategies being brought into the next generation. A smaller population size will maintain a higher fitness but may not attain the best case over the course of the generations. This is the exploration versus exploitation dilemma. Having too large a population size will reduce the overall fitness of the population, but may lead to better results over time.

<p>| Population Size | 5, 10, 20 |</p>
<table>
<thead>
<tr>
<th>Strategy Length</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generations</td>
<td>10</td>
</tr>
<tr>
<td>Maturation factor</td>
<td>0.3</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Table 6 Parameters for the Population Size test**

The results show that the average affinity increased over the period of ten generations. This means that the adaptation process performed as it should have. The rate of adaptation slowed down after each generation as the AIS adapted to the strategies. This is understandable as the AIS is easily able to find better strategies earlier in the adaptation process than after successful strategies have already been found.

![Population Size Adaptation](image)

**Table 7: The average affinity from each of the ten generations run by the AIS on each population size**

Table 7 shows that all three population sizes were comparable in their increase in average over the course of the 10 generations. The population size of 10 and 20
performed marginally better than the population size of 5. The most likely reason for this is that the larger population sizes are more forgiving to strategies that do not perform as well. In the MetaDynamics phase, when the strategies are being removed due to poor performance, a larger population size can retain more lower scoring strategies than a smaller population size.

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.41</td>
<td>0.42</td>
<td>0.42</td>
</tr>
<tr>
<td>Std</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
</tr>
<tr>
<td>Min</td>
<td>0.17</td>
<td>0.16</td>
<td>0.17</td>
</tr>
<tr>
<td>Max</td>
<td>0.55</td>
<td>0.57</td>
<td>0.57</td>
</tr>
<tr>
<td>Linear reg</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
</tr>
</tbody>
</table>

*Table 8 Compiled results from the Population Size tests*

Table 8 shows that the total average affinity for each population size is almost the same. As the population size increases, the performance of the AIS increases slightly in our test environment. This result may be different for other game environments.

**Change in Maturation Factor**
If a selected strategy has a high enough affinity to make it into the next generation, it will be evaluated again as part of the current population of strategies for the next generation. In these tests the result of an engagement of two of the same strategy can change from one generation to the next depending on how it performs in its engagement. If a strategy performs better in an engagement in following generations then its affinity increases based on the score of the engagement and the Maturation Factor. If a strategy performs worse, then its affinity will decrease in the same way. Table 9 shows some examples of how the Maturation Factor has an effect on the calculated affinity.

<table>
<thead>
<tr>
<th></th>
<th>Example 1</th>
<th>Example 2</th>
<th>Example 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affinity Generation 1</td>
<td>0.75</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>
With a high maturation factor, a positive result from an engagement will lead to a higher rise or fall in affinity, whereas a lower maturation factor will produce a lower change.

This is not unlike the stability versus plasticity dilemma, where new elements are to be learned while still retaining previously learned elements (Abraham & Robins, 2005). The Maturation Factor can enable the rate at which old strategies are retained and new strategies are learned.

Unlike a biological immune system where the affinity between a specific antigen and antibody does not change over time, in a game environment, the result from one engagement to the next is generally not exactly the same. In a deterministic game the affinity would remain the same, but most modern games have a lot of variables that may come into play from one engagement to the next. So there could be variances in the strategies involved that may produce a slightly different outcome to an engagement, however we do not want the last result to be the one that determines the affinity. By employing a maturation factor we can augment the computer strategy that is currently active to reflect these variances and the inherent unpredictability that a human player might introduce into an engagement.

<table>
<thead>
<tr>
<th>Maturation Factor</th>
<th>0.2</th>
<th>0.2</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result Generation 2</td>
<td>0.5</td>
<td>0.75</td>
<td>0.5</td>
</tr>
<tr>
<td>Calculation</td>
<td>0.75 + (0.2 * (0.5 - 0.75))</td>
<td>0.5 + (0.2 * (0.75 - 0.5))</td>
<td>0.5 + (0.5 * (0.5 - 0.5))</td>
</tr>
<tr>
<td>Affinity Generation 2</td>
<td>0.70</td>
<td>0.55</td>
<td>0.5</td>
</tr>
</tbody>
</table>

**Table 9 Examples of some average affinity calculations**

Table 10 Parameters for the Maturation Factor test

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy Length</td>
<td>5</td>
</tr>
<tr>
<td>Generations</td>
<td>10</td>
</tr>
<tr>
<td>Maturation factor</td>
<td>0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.2</td>
</tr>
</tbody>
</table>
A lower Maturation Factor causes a much more gradual increase in the average affinity compared to a higher Maturation Factor. This would be beneficial in a game environment if the rate of adaptation needed to be slowed, maybe based on the game level or the skill of the human player. A higher Maturation Factor has a profound effect on the adaptation rate right from the first generation. However, although the affinity of a Maturation Factor of 0.9 is much less of a variance in average affinity than 0.1, there is a much greater
variance of the strategies within the current population. The reason for this is that if a strategy does not perform well, then its affinity can immediately drop to a point where it is removed from the strategy collection.

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.23</td>
<td>0.34</td>
<td>0.43</td>
<td>0.47</td>
<td>0.52</td>
<td>0.55</td>
<td>0.56</td>
<td>0.61</td>
<td>0.62</td>
</tr>
<tr>
<td>Std</td>
<td>0.11</td>
<td>0.13</td>
<td>0.13</td>
<td>0.12</td>
<td>0.10</td>
<td>0.10</td>
<td>0.07</td>
<td>0.06</td>
<td>0.05</td>
</tr>
<tr>
<td>Min</td>
<td>0.06</td>
<td>0.11</td>
<td>0.17</td>
<td>0.23</td>
<td>0.29</td>
<td>0.32</td>
<td>0.40</td>
<td>0.47</td>
<td>0.49</td>
</tr>
<tr>
<td>Max</td>
<td>0.36</td>
<td>0.50</td>
<td>0.57</td>
<td>0.58</td>
<td>0.60</td>
<td>0.63</td>
<td>0.63</td>
<td>0.67</td>
<td>0.66</td>
</tr>
<tr>
<td>Linear reg</td>
<td>0.03</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.03</td>
<td>0.03</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Table 12 Compiled results from the Maturation Factor tests

Table 11 shows that a higher Maturation Factor leads to a higher average affinity in a shorter period of time. In a game scenario this might be necessary in the skill of the player is high. In a player has a lower skill level then a more gradual adaptation process could be preferable. In this case, a lower Maturation Factor may be more desirable. Another scenario where the Maturation Factor might have pertinence is changing in difficulty levels of a game, where the Maturation Factor increases in proportion to the game difficulty.

**Change in Mutation Rate**

The Mutation Rate is the variable which controls the amount of diversity is introduced when a strategy undergoes mutation. A strategy with a lower affinity will mutate more than a strategy with a higher affinity. This is the main component of the AIS that contributes to how rapidly it can find an optimal solution.

The mutation functions that are used for this experiment are:

If the affinity is less than the mutation rate then the strategy will switch the position of two of its tactics, i.e. ABCDE might become BACDE.
If the affinity is less than half the mutation rate, then the strategy will replace one of its tactics with a new randomly selected tactic from the tactic library.

The strategy will undergo mutation function 1 if the mutation rate and affinity pass a certain threshold. If this threshold is reached, then a further check is performed to evaluate whether the strategy will also undergo mutation function 2.

<table>
<thead>
<tr>
<th>Population Size</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy Length</td>
<td>5</td>
</tr>
<tr>
<td>Generations</td>
<td>10</td>
</tr>
<tr>
<td>Maturation factor</td>
<td>0.3</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9</td>
</tr>
</tbody>
</table>

*Table 13 Parameters for the Mutation Rate test*
Table 14 compares the different changes in average affinity for each of the ten Mutation Rates tested. A mutation rate of 0.1, 0.2, 0.3 and 0.4 show very little difference in their performance, each having a similar rate of increase of average affinity over the ten generations. As the Mutation Rate increases, so does the rate at which the AIS achieves a better average affinity. An interesting point to note is that the Mutation Rates of 0.8 and 0.9 also have a higher average affinity at generation ten than the rest of the Rates. The reason for this is as these Rates were able to achieve a high average affinity very quickly,
the AIS was able to spend the rest of the generations improving on these strategies that generated the high affinities.

<table>
<thead>
<tr>
<th></th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.43</td>
<td>0.43</td>
<td>0.43</td>
<td>0.46</td>
<td>0.48</td>
<td>0.50</td>
<td>0.51</td>
<td>0.59</td>
<td>0.62</td>
</tr>
<tr>
<td>Std</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
<td>0.13</td>
<td>0.11</td>
<td>0.11</td>
<td>0.08</td>
<td>0.07</td>
<td>0.06</td>
</tr>
<tr>
<td>Min</td>
<td>0.18</td>
<td>0.18</td>
<td>0.17</td>
<td>0.21</td>
<td>0.25</td>
<td>0.25</td>
<td>0.32</td>
<td>0.41</td>
<td>0.47</td>
</tr>
<tr>
<td>Max</td>
<td>0.57</td>
<td>0.55</td>
<td>0.57</td>
<td>0.58</td>
<td>0.56</td>
<td>0.58</td>
<td>0.57</td>
<td>0.62</td>
<td>0.66</td>
</tr>
<tr>
<td>Linear reg</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.03</td>
<td>0.03</td>
<td>0.02</td>
<td>0.02</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Table 15 Compiled results from the Mutation Rate tests**

One drawback to having a high mutation rate is that when adapting to multiple strategies, this can affect the AIS's ability to retain strategies or fragments of strategies that were successful in previous encounters.

**Change in Maximum Strategy Length**

This experiment tests the effect that different strategy lengths have on the average affinity. The strategy length is the maximum number of individual tactics that can make up a strategy. For testing purposes, the individual tactics for our AIS contain all the required functions to act as an autonomous entity. So even a strategy length of one can perform all the functions required to provide a competitive opponent. Having a longer strategy length increases the amount of tactics and decreases the predictability of the strategy, as the complexity of the strategy increases as more tactics are added.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>10</td>
</tr>
<tr>
<td>Strategy Length</td>
<td>1,3,5,7,9</td>
</tr>
<tr>
<td>Generations</td>
<td>10</td>
</tr>
<tr>
<td>Maturation factor</td>
<td>0.3</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.2</td>
</tr>
</tbody>
</table>

**Table 16 Parameters for the Strategy Length test**
Table 17 The average affinity from each of the 10 generations run by the AIS with a strategy length of 1, 3, 5, 7 and 9

Table 17 shows the average affinity of each of the five different strategy lengths. The results show that there is little to differentiate between them. There are small variances over each of the generations but not enough to establish a trend. Probably the main contributor to this is the fact that the tactics are complete and functional units, so adding multiples will not have much of an added effect.
<table>
<thead>
<tr>
<th>Max</th>
<th>0.56</th>
<th>0.53</th>
<th>0.55</th>
<th>0.47</th>
<th>0.55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear reg</td>
<td>0.04</td>
<td>0.04</td>
<td>0.04</td>
<td>0.03</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 18 Compiled results from the Strategy Length tests

For this thesis, the AIS tactics were very coarse grained, i.e. each tactic could be in itself a strategy. For more complex scenarios where strategies are made up of many more tactics further testing would be required.

**Conclusion**

The parameters that can augment the behaviour of an AIS are the Population Size, Maturation Factor, Mutation Rate and Maximum Strategy Length. These parameters could be present in all AIS implementations and are not specific to this thesis. The Maturation Factor and Mutation Rate have a much more noticeable effect on the AIS than do the Population Size or Maximum Strategy Length. The Maximum Strategy Length that was tested on in this Chapter was low, but in other implementations this number might be much greater in order to provide strategies that are much more complex and dynamic.
Chapter 5 - How the AIS recalls previous encounters

In Chapter 5 we investigate the ability of the AIS to learn strategies and use that learned knowledge when the same or similar strategies are encountered again. This is one of the major features of an AIS, and drawn from the biological immune system's ability to recognize previously encountered strategies and adapt quickly to counter them. The encounters between antigens and the biological immune system could be compared to the encounters between the human players' strategies and the AIS. The benefit that this brings to adaptation and learning in computer games is that the AIS can be trained offline against multiple strategy types, then, when faced with a human opponent it does not need to relearn a counter-strategy, only adapt one that it has previously learned. If the AIS can remember and adapt quicker to previously encountered strategies, then this makes a strong case for the AIS as a strategy generation mechanism as the opponent must continually adapt as the AIS adapts, and not just revert to earlier successful strategies.

Also, there is a benefit into not allocating computing resources into rediscovering counter strategies that have already been discovered previously.

How the AIS remembers past encounters

The ability of a biological immune system to be able to remember what antibodies have been successful in combating antigens which have invaded the system is crucial to its operation. If the immune system did not remember which antibodies were successful, it would be required to search again for an appropriate countermeasure every time an infection occurred, even one that had been previously encountered and resisted. This is the reason why inoculations are effective in preventing us from catching certain diseases. Our bodies are presented with a mild or inactive form of a pathogen (e.g. a virus), and our immune system is able to develop antibodies that are effective against it. Then when the
real virus attacks our body, our immune system already has the required antibodies to fight off the infection and what prevents us from getting sick.

The way the biological immune system keeps a record of successful antibodies is through memory cells (Schindler, 1991). These memory cells are a subset of B and T cells that are produced during an immune response. These cells have a much longer lifespan than regular immune cells and are responsible for the immune system being ready for an invasion of the same pathogen that elicited the response in the first place.

The way AISs remember previous encounters in done in a similar manner. When the AIS is activated, depending on the level of affinity, the successful countermeasure is recorded. If a better countermeasure is discovered during the course of the response then this is also recorded. After a number engagements the AIS has produced the best countermeasures that it could find within the allocated time or generation count.

This approach could be likened to a computer game opponent being inoculated against the human players that will be playing the game. This experiment will use the same three opponent Robots from Chapter 4, Toorkild, Thorn and WaveShark. The AIS will be introduced to each Robot's strategy, and will build up a memory of the best countermeasures for each strategy. The result should be a collection of strategies that are able to adapt with a quicker response time than when encountering the strategy for the first time.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>10</td>
</tr>
<tr>
<td>Strategy Length</td>
<td>10</td>
</tr>
<tr>
<td>Generations</td>
<td>10</td>
</tr>
<tr>
<td>Maturation factor</td>
<td>0.4</td>
</tr>
<tr>
<td>Mutation Rate</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Table 19 Parameters for the Offline Learning and subsequent Adaptation

Table 19 describes the parameters that we will use for the test. These parameters were chosen as they provided what could be considered a "fair" setting for the AIS with reasonable adaptation speed and the ability to retain strategies from previous encounters.

Table 19 Parameters for the Offline Learning and subsequent Adaptation

<table>
<thead>
<tr>
<th></th>
<th>Toorkild</th>
<th>Thorn</th>
<th>WaveShark</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>ACIHGFH</td>
<td></td>
</tr>
<tr>
<td>IJGAAJFDB</td>
<td>AIAIAC</td>
<td>AJGBIJFDA</td>
<td></td>
</tr>
<tr>
<td>AIAIAC</td>
<td>HIAHCFG</td>
<td>IJFAAJBDG</td>
<td></td>
</tr>
<tr>
<td>HCHAIFG</td>
<td>FI</td>
<td>IJGBAJFDA</td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td>IJGAAJFDB</td>
<td>DIGAAJFJB</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4 The average affinity recorded when training against multiple strategies. Generations 1-10 against Toorkild, Generations 11-20 against Thorn and Generations 21 - 30 against WaveShark

Figure 4 shows how the average affinity of the AIS changes as more strategies are learned. Initially, when the AIS is trained against a single strategy, the average affinity increases as the AIS learns the best counter strategies against the opposing strategy.

When more strategies are learned, the average affinity drops, as the AIS adapts to the new strategies. The final set of strategies are the ones that will make it into the game, and can be considered to be a good performer when those same or similar strategies are encountered again.
Table 20 The top ten strategies from each of the strategies encountered ordered by effectiveness

<table>
<thead>
<tr>
<th>GDJJEFBHI</th>
<th>GHHHCEDH</th>
<th>IJFAAJGDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DJGEDG</td>
<td>H CIAHFG</td>
<td>IJGAAJFDB</td>
</tr>
<tr>
<td>GEJHDFBJI</td>
<td>HCAHIFG</td>
<td>F</td>
</tr>
<tr>
<td>GHHHCEDH</td>
<td>DJGEDG</td>
<td>AHICDFG</td>
</tr>
<tr>
<td>GEJJDFBHI</td>
<td>GEJJDFBHI</td>
<td>HCAHIFG</td>
</tr>
</tbody>
</table>

Table 20 shows the final strategies that were generated from each of the strategies encountered. The strategies under the "WaveShark" column are the cumulative strategies from the learning process and are the strategies that will be used to evaluate the performance of the adaptation speed for the AIS against all three strategies.

Time taken to adapt to changing opponent strategies

One of the goals of this thesis is to show that an adaptive approach can be achieved within reasonable tolerances of observable performance. In a game environment, when the AIS has adapted to the human player's strategy, the logical next step is for the human player to try a new strategy. There should be little noticeable effect when the opponent changes tactics forcing the AIS to adapt. It is understandable that when the human player changes strategies that they either expect their new strategy to perform well or poorly against the AIS strategy. If it performs poorly, then it is understandable that the human player would disregard that strategy and continue looking for a strategy that provided more positive than negative results. This will cause the AIS to learn and adapt to the new strategy, however there may be a drop in performance as the AIS adapts to the new strategy.

The main criteria that a computer opponent has to satisfy is that it provides a challenge to the human player. When combined with an AIS, the computer opponent should provide ongoing and intuitive challenges as the human player learns and adapts to the computer
opponents' strategies. It would be futile to adapt to a strategy only for the human player to revert back to a previous tactic, the strategy for the human player would be then: Use strategy A until the computer adapts, then switch to strategy B. When the computer adapts to strategy B, switch back to strategy A.

![Adaptation with previous encounter](image)

**Table 21** Adaptation of the Toorkild strategy when the strategy has been previously encountered against when it has not been previously encountered

As discussed earlier, when an AIS first encounters an opposing strategy, there is a period of lower than average performance as the AIS adapts to the new strategy. If the AIS is encountering a strategy that it has already encountered at some point in the past, this period should be greatly reduced and the AIS should adapt much more rapidly to the strategy compared to when it was encountered for the first time. Table 21 shows that adaptation is much quicker, and achieves better performance when the AIS has encountered the opposing strategy in a previous encounter. Even though the previous encounter had since been augmented by the two subsequent encounters with the Thorn
and Waveshark robots, it was still able to adapt at a much quicker rate than having no previous encounter.

To determine if it is always the case that the AIS adapts quicker when encountering strategies that it has previously encountered we need to expand the number of encounters and perform the test on a larger scale.

For the second test we train the AIS using two separate encounters, the first against the Toorkild Robot and the second against the Thorn Robot. After the training there is second encounter between the AIS and each of the opposing Robots. Unlike the previous adaptation test, this test is designed to compare the first encounter of the AIS with a strategy and then evaluate what happens when that strategy reappears. This would be similar to the human player switching back to a previously successful strategy after the AIS had adapted to it and countered it.

The test will compare the first encounter with the Thorn strategy against the second encounter with the Thorn strategy. The Thorn strategy was chosen above the Toorkild strategy to mitigate any variance in the AIS adapting against an initial strategy (which is Toorkild in this case). To evaluate the performance of the adaptation, the number of generations it takes in each encounter to reach the average affinity was recorded.
The test was run through 99 encounters and at the end of each encounter the number of generations that it took for the AIS to reach or surpass the average affinity was recorded. This was compared with the turns to reach the average affinity from the first encounter. The generations that the AIS took to reach the average affinity for the encounter are shown in Figure 5 and Figure 6. It is clear that in the second encounter the amount of generations taken to reach the average affinity are less than the first encounter. During the first encounter most of the population resides above the median stating that it took between 2 and 5 generations before the average affinity was reached. In comparison, during the second encounter fully half of the population resides on or below the median,
with the average affinity being reached in 1 to 4 generations. There are outliers in both Figure 5 and Figure 6 which represent encounters where it took a large amount of generations to reach the average affinity. The innate ability of the AIS to remember and quickly adapt to previously encountered strategies is a property that is something that currently is beyond the capabilities of scripted techniques. And it is these characteristics that could make a strong argument as to the benefits of having an adaptable strategy generation over a static scripted one.

<table>
<thead>
<tr>
<th></th>
<th>Encounter 1</th>
<th>Encounter 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test count:</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>Median</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Minimum</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Maximum</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>First quartile</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Third quartile</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Interquartile Range</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 22 Results of the amount of generations taken to reach the average affinity.

Table 22 shows that although the results are quite similar, there is an improvement in adaptation speed in the second encounter over the first encounter. This is most obvious comparing the first and third quartiles, where the second encounter is a full generation quicker than the first encounter.

**Factors that can influence the recollection of successful strategies**

There are some factors that can influence the AIS's ability to persist successful strategies through successive encounters; an encounter being when the AIS meets an opposing strategy:

**Population Size:** The population size is probably the most important attribute which enables the AIS to persist successful strategies through successive encounters. To recap,
the population size is the count of the collection of strategies that the AIS can work with in any given generation. A larger population size enhances the ability of the strategies from earlier encounters to remain either partially or fully intact after subsequent encounters against different opposing strategies.

With a smaller population size it is entirely possible that all successful strategies are augmented to be successful against the current encounter, and there are no successful strategies remaining that were produced from previous encounters. Due to how this AIS implementation works through mutation and maturation, there may still be components of strategies that were successful from previous encounters remaining in the resulting strategies from the current encounter. It is this that is essential for the AIS to be able to have a memory of what countermeasures were effective in previous encounters. Table 20 shows how the strategies have augmented over three successive encounters, and it is clear that there are still components of strategies that were successful in the first two encounters against Toorkild and Thorn in the strategies that were successful in the final encounter against Waveshark.

**Maturation Factor:** The maturation factor has an effect on the ability of previous successful strategies to be maintained through subsequent encounters. When the AIS employs a previously successful strategy that may not be suitable as a counter-strategy for the current opposing strategy, the affinity will be lower than for the previous encounter. A high maturation factor will reduce this successful strategies affinity to a point where it is removed from the current AIS's strategies and will not be included in the successful strategies from this encounter. A low maturation factor will not be as severe on lesser performing successful strategy and may remain to the end of the current
encounter. If the maturation factor is too low then subsequent encounters will take much longer in adapting to the new opposing strategy.

**Mutation Rate:** A high mutation rate means that a successful strategy in the AIS's current strategies has a greater chance of mutation, especially if it does not perform as well against the current opposing strategy. This is proportional to the level of affinity, and a strategy that is not as competitive in a current encounter will have a lower affinity and hence a greater degree of mutation will occur. If the mutation performs well against the current opposing strategy it is likely that this mutated strategy will also perform well against the previous encounter's opposing strategy, thus making it an excellent candidate for following encounters against similar or the same opposing strategies.

**Number of Encounters:** As the AIS encounters more and more strategies, previous successful strategies that have not been encountered for a long time start to dissipate from the AIS's memory. This is a result of the generalization of the AIS's successful strategies, which can be seen in Figure 4. As the AIS learns to counter more recent opposing strategies, previous successful strategies dissipate or are incorporated into more current successful strategies.

**Conclusion**
This chapter has shown that the AIS is clearly capable of remembering successful strategies from previous encounters, and when the strategies or ones like them are encountered again the rate of adaptation of the AIS is much faster. This ability is something that is beyond the scope of current scripted techniques which will always perform the same way against any strategy that the human opponent uses. As in Chapter
4, the AIS parameters also have an effect on the rate of adaptation when previously encountered strategies need to be dealt with again.

As the AIS does not have to relearn and readapt against previously encountered strategies, this could provide a semblance of intelligence to the human player if they decide to revisit a previously successful strategy. Also, as the AIS’s strategies are not so specific to target only one strategy, if the AIS encounters strategies that are similar to ones it encountered previously, then the rate of adaptation should be greater than encountering a completely novel strategy.
Chapter 6 - Analysis of the test results
In this chapter the results described in the previous chapters are assessed and we
determine what the effect on the adaptation process the different parameters exhibited.

We also look at the level of control that is available after the strategies have been
generated, and how easy or difficult it is to follow the strategy generation process and see
how the strategies evolve and adapt. These results will show how the initial goals were
achieved and how the observability, control and adaptability are all attained through the
AIS implementation. The developmental progression of the AIS implementation is also
described, and how the final version is the one that is most suited for this type of study.

We also look at Robocode as a suitable environment for running automated tests and
whether this platform could be used again for similar studies.

Evaluation of the results
With the implementation of the AIS in this paper the following are the results for each of
the tested parameters:

Population size: In the AIS the population size is the amount of strategies that are active
in any given generation. These strategies are a subset of the previous generations’
successful strategies. If the successful strategies number less than the population size,
random strategies are added.

During the adaptation process, a smaller population size seems to perform slightly worse
than a larger population size. The most likely reason for this is that as the population size
increases, so too does the diversity of the strategies that the AIS can use for the current
generation. During the Metadynamics stage of the AIS process the lowest scoring
strategy is removed to be replaced with a previously successful strategy from the pool or
a new random strategy. With a population size of 5 this accounts for 20% of the entire population, whereas with a population size of 20, one strategy only counts for 5% of the population. The drawback of having a smaller population size is that there are less strategies which will undergo mutation during the Affinity Maturation phase. This will inhibit the AIS's ability to find more suitable strategies.

**Mutation Rate:** Whether a strategy mutates during the Affinity Maturation phase is dependent on two factors. It's level of affinity and the mutation rate. A high affinity and a low mutation rate means that there is little chance that the strategy will mutate, whereas a low affinity and high mutation rate means that strategy and its clones will undergo substantial mutation.

In this AIS implementation, the average affinity of successful strategies grows in proportion to the mutation rate. A higher mutation rate results in more successful strategies as there are more clones and mutated strategies present. While this may lead to better strategy generation, it may inhibit the AIS's ability to remember past successful strategies.

**Maturation Factor:** In the case of Robocode and other environments, where the same parameters may not give the same results during subsequent engagements between the same strategies, the maturation factor is used as a limiting device. The selected AIS strategy might perform well for a certain number of engagements, but then perform poorly on the next. This is not dissimilar to a human player playing a game where they get a lucky result from an engagement with a computer opponent. However in the case of
the AIS, a poor result could lead to that strategy being removed from the successful strategies collection.

The maturation factor is primarily responsible for ensuring inconsistent engagements do not skew the affinity from previous engagements with the same strategy. A higher maturation factor will lead to much greater changes in affinity from generation to generation for a given strategy, while a lower maturation factor will lead to smaller increases or decreases in changes of affinity with regard to the changes in affinity from one engagement to the next. The maturation factor is not a traditional component of AIS, but was used in these tests to prevent fortunate results from having too much of an impact on the AIS's learning. However the maturation factor is not only applicable to strategies that already exist in the AIS's successful strategy collection. If a strategy has a poor result initially, with a high or low maturation factor, the chances are that it will never make it into the successful strategy collection.

From the results in Chapter 4, as the maturation factor increases, so too does the rate at which the AIS reaches convergence against a single strategy. The higher maturation factors were able to attain a higher average affinity within 10 generations, compared to lower maturation factor values which, after 10 generations, the average affinity was still rising.

**Maximum Strategy Length:** The results show that, for this AIS implementation, having a larger strategy results in a decrease in performance. This may not necessarily be the case for other implementations as the strategies are totally dependent on the environment in which the AIS is implemented and the discretion of the implementer. For these tests,
each element in the strategy string, represents one complete robot tactic, so that a strategy of length one would still produce a functional robot. A robot with a strategy length greater than one would produce a robot whose strategy is a combination of all the tactics that make up the strategy. So it seems that in this implementation, the greater complexity of larger strategies has a detrimental effect on how the robot performs.

**Control over the generated results**
Each strategy has a group of tactics corresponding to a letter in the strategy. These tactics are the actions that the in game character will take as determined by the game developers. If the developers want more control over the character's behaviour, then the tactics can contain a full set of contiguous actions, similar to a full script in a scripting environment. Control is taken from the AIS to generate diverse strategies and given to the developer to maintain a much higher level of control over the character's behaviour. As in our experiment, if the tactics are going to be combined, they must have the ability to work together or some method of subsuming previous tactics that make up a strategy.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0.57</td>
</tr>
<tr>
<td>HHBDFBB</td>
<td>0.56</td>
</tr>
<tr>
<td>FIC</td>
<td>0.54</td>
</tr>
<tr>
<td>IC</td>
<td>0.54</td>
</tr>
<tr>
<td>FIHDF</td>
<td>0.5</td>
</tr>
<tr>
<td>IJFFEJBE</td>
<td>0.47</td>
</tr>
<tr>
<td>EDHEFB</td>
<td>0.41</td>
</tr>
<tr>
<td>GEJBF</td>
<td>0.36</td>
</tr>
</tbody>
</table>

*Table 23 The results generated against the Toorkild robot after 10 generations, with a mutation rate of 0.2, maturation factor of 0.1 and a maximum strategy length of 10. From the results, Strategy G is the most successful.*

Table 23 shows that the end result is purely a collection of tactics compiled into successful strategies, with an associated affinity or level of performance of that strategy as a result of previous encounters. A game developer might like to develop the strategies
manually and use them as a starting point for the AIS adaptation. This level of control is something that is provided by an AIS implementation above and beyond other methods of learning and adaptation. This is one of the main reasons that an AIS implementation can bridge the gap between having the control of scripting yet the benefits of an adaptive opponent. Also, depending on the complexity of the implementation, strategies can be created manually if required, and then allow the AIS to fine tune them through training against opponent strategies.

As discussed in Chapter 3, we recommended that the functionality for mutating and evaluating the strategies can be abstracted from the AIS implementation. This allows the game developers to implement their own mutation and evaluation functionality, much of which is game specific.

<table>
<thead>
<tr>
<th></th>
<th>Mutation with high affinity</th>
<th>Mutation with low affinity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>ABCDEFG</td>
<td>ABCDEFG</td>
</tr>
<tr>
<td>Round 1</td>
<td>ABCDEGF</td>
<td>ABCDFEG</td>
</tr>
<tr>
<td>Round 2</td>
<td>ABCDEG</td>
<td>DBCDFEG</td>
</tr>
</tbody>
</table>

*Figure 7 The mutation process as implemented in the Robocode AIS during Affinity Maturation for the strategy ABCDEFG. Low affinity produces more mutation than high affinity.*

As the mutation and evaluation implementations are abstracted from the core AIS system, developers have full control over the level of mutation or how the generated strategies are evaluated. As well as this, the output from each evaluation of the AIS can be recorded to see if the performance of the AIS is as expected. (see Appendix A)

**AIS Implementation Evaluation**

Throughout the course of this investigation, the AIS implementation has undergone several iterations of refinement and refactoring. Initially the AIS algorithm was incorporated within the robot itself, making it an autonomous learning agent. This meant
that it was the robot itself that evolved and adapted in accordance with its own performance results from engagements. While this worked quite well, and probably is a more appropriate implementation in some cases, for the purpose of these experiments it proved to be too cumbersome and hard to transition what one robot had learned to another.

The second implementation extracted the AIS from the robot. This meant that there was one AIS that acted as the provider of strategies for multiple robots over concurrent engagements. This model represents better how a biological immune system works, the robots representing the immune cells and the resultant affinities from engagements being processed by a central system. The second major change was in the initial implementation there was only one set of strategies. This set was constantly changing based on the performance of the robot. The second implementation brought in the concept of a separate collection of "successful" strategies that evolved at a much slower pace and was used as a source of strategies for the robots current strategy pool. This method proved to be a lot more conducive to running multiple subsequent encounters, and gathering the results for processing and evaluation, and enabled the AIS to remember and adapt not just against the current opposing strategy, but to learn over the course of many encounters.

The third implementation was a refinement over the second implementation. With the processing and evaluation of the generated results being much easier, it was clear where certain components of the AIS implementation needed to be refined or reworked. E.g. the procedure for performing the metadynamics in the second implementation proved to be too severe in its extraction of poorer performing strategies. The result of this was that the
memory of the AIS was drastically reduced and was not able to recall strategies that had
performed well during previous encounters when that strategy was encountered again.
Another example of refinement was in the procedure that performed the mutation. In the
biological Immune System when the B Cells interact with an antigen, they undergo
hypermutation, the rate of which is based on their affinity. In the second implementation,
this hypermutation was subdued, resulting in not so many clones of higher affinity
strategies being produced. In the third implementation, the mutation rate had a more
profound impact on the amount of clones of higher affinity strategies produced.

In practice, a game which has distinct in-game opponents or personalities, should
implement the same model as the first implementation, where each opponent has its own
implementation of an AIS. Another implementation could be in games where there is a
persistent environment, as in massively multiplayer online games, where, over time the
AIS implementation learns and adapts as more and more players interact with it. In a
game where there are multiples of the same opponent, like an RTS game, the second
model is much more appropriate.

**Robocode as a platform for adaptation and automated testing**
The Robocode environment offers an excellent platform for strategy generation as it has
almost limitless possibilities for strategy compilation. This implementation used quite a
large collection of commands for each tactic, but the same methodology can be used to
create much simpler tactics, and hence much more complex strategies.

As Robocode could be considered quite a mature platform, having undergone many
generations of upgrades and maintenance, and as there is currently a large community of
active developers, the likelihood that the generated results are misleading or
misrepresentative of the performance of the strategies is very remote. The Robocode engine also supports the ability to be run automatically, through command line tools or through code. This makes automated testing and result gathering require a lot less effort than if automation was not possible. All of the results in this thesis were generated through the automated running of the Robocode engine.

**Conclusion**
Each of the AIS parameters affect a unique component of the AIS implementation, and together they control how the entire system functions. For the implementation described in this thesis, the tests show that the change in the Maturation Factor had the most profound impact on the system. This was followed by the Mutation Rate, which increased the rate of adaptability as its value increased. The Population Size and Maximum Strategy Length parameters did not have much of an impact on the AIS's performance, however as stated above, this may not be the case for other AIS implementations.

Throughout the tests the AIS showed that it could consistently adapt to opponents strategies, whether it had encountered that strategy before or not. The rate and mode of the adaptation can be altered by variations in the AIS's parameters, which gives developers a means of control to reach the desired functionality. As in this implementation, the results of the AIS learning and adaptation process can be visualized through inspecting the chosen strategies during each generation for every encounter that the AIS makes against an opposing strategy. This is extremely useful in understanding what direction the AIS is taking against any particular strategy at any point in time. This level of observability is deemed necessary for the success and uptake of any new adaptation implementation in modern computer games.
The final implementation of the AIS has achieved all the goals in terms of adaptability, control and observability to make it a viable means to bring an adaptive component to computer games, and could be a good alternative to scripted approaches that are still prevalent in most contemporary computer games.
Chapter 7 - Conclusion and further work

The main goals of this study were to determine if an AIS controlled computer component satisfied the three criteria identified previously to be considered as a potential next step in the evolution of adaptable computer opponents. The first criteria was observability, or that the process of generating the counter strategies, could be clearly followed. The second criteria was control, or the ability of the game developers to be able to augment to a certain degree the results of the AIS learning and adaptation process. Finally, and most importantly, the AIS could adapt and produce strategies that were competitive and comparable to a scripted opponent. This chapter will evaluate whether these goals have been achieved,

AIS as a suitable vehicle for learning and adaptation in games

Previous methods for adaptation in games tend to be tightly coupled to the mechanics of the game itself (Spronck, Sprinkhuizen-Kuyper, & Postma, 2003). An AIS can be implemented in any game where there is an encounter between the human player and a computer controlled opponent. There are no dependencies on how the game is implemented, or what the success criteria are for player success. Separating the adaptive logic from the game implementation should be considered a driving factor for the development of an adaptive component of a game. There are two reasons for this:

The same logic can be tried and tested across multiple games, thereby increasing exposure and confidence.

The implementation does not have to be constructed from scratch each time it is used.

The performance of adaptation in games has been widely discussed, and determined to be one of the major reservations to the inclusion of AI techniques in modern video games
(Spronck, Sprinkhuizen-Kuyper, & Postma, 2003), (Manslow, 2002). The AIS implementation is not a drain on resources as the main constituent of what the AIS needs to adapt are encounters within the game itself as these encounters will happen whether an AIS is learning from them or not. However, a larger maximum strategy length value and population size value, can have an effect on performance as more resources are consumed in performing the AIS adaptation and learning. The larger these values are the more searching has to be done by the AIS, the trade off being that adaptation and learning become more accurate. Although, as the results in Chapter 4 have shown, a larger maximum strategy length is heavily associated with how the tactics are implemented within the game. Even without the learning or adaptation component, an AIS implementation is a straight forward and manageable method for strategy variation in computer controlled opponents. If a game required some dynamism, the AIS could be used strictly to serve up the most appropriate strategy in its repertoire without the unpredictable learning component attached. Without the learning component developers can combine tactics that are already scripted and use the AIS engine to generate the strategy.

For the AIS to be effective within a video game, there must be some amount of engagement between the human player and the computer. The AIS usually determines how effective its current strategy is through a series of engagements, and subsequent engagements will show the adaptation from the previous engagements. Games with many engagements will benefit more from having an adaptive strategy generation mechanism, as there will be many more opportunities for the system to learn and adapt.
Another benefit to both the game developer and end user is the ability of the adaptive AIS to naturally learn to avoid exploits or unintended tactics within the game. Quite often in modern computer game, exploits are uncovered during the course of play that are unanticipated during game development. Initially, to the end user, these would seem enjoyable. They have outwitted the game and discovered a strategy that the computer player has no recourse for. Traditionally if these exploits become known to the game developers, the usual way to fix them is to patch the game so that the piece of code, or lack thereof, that made the exploit available has been rectified. The AIS, by its nature, will avoid strategies that are resultant from exploits, as they will generate low scoring results. Also the end user, having witnessed the computer opponent realise the exploit and compensate, is faced with a new and more challenging opponent. In the case of a game using an AIS to direct the computer opponents tactics and strategies, exploits like these need not be a problem.

From a technological perspective, the inner workings of the AIS are not overly complex, and do not require any specialised skills or dedicated resources to develop. Chapter 3 clearly outlines all components required for a comprehensive AIS implementation, each of which have a basic structure, but combined become a powerful adaptation tool.

**Offline learning**

There is potential for the AIS to perform the majority of its learning offline. During game development the AIS can be trained against automated computer opponents, human opponents or other AIS enabled opponents. Once the training is complete, the AIS should have a sufficient repertoire of strategies to be able to recognize and adapt to most strategies that the end user will employ.
When the game is then played by the end user, if the user chooses a strategy that the AIS has encountered previously during its offline learning, the rate of adaptation will be greater than if the strategy had not been encountered before. As has been demonstrated in Chapter 5, the AIS adapts quicker to strategies that it has encountered previously, and as a result the end user experiences less of a degradation in performance of the opponent as the AIS adapts to the new strategy.

AIS controlled opponents that have been trained offline can be evaluated before the game is released. As AIS generated strategies are easily modified by game developers after the learning has taken place, the strategies can be tweaked or augmented to suit the environment of that character in the game. If the character is located within a forest, the developers might augment the AIS with strategies that favour using trees as cover. The effect that this would have on the game is that the AIS would adapt to its best previously learned strategy to counter the human player's current strategy. This might be more desirable in terms of removing any unpredictability from the AIS control system, however, it also removes the ability of the AIS to learn, if none of the previously trained strategies are sufficient to counter the current opponent.

**Distributed learning**

One of the main features of using an AIS as a learning component in games is that the more engagements the AIS encounters, the better it is at being able to learn new counter strategies. With modern games becoming more and more distributed, it is certainly possible for an AIS to be used by multiple instances of a game running online.
It would be plausible that a single instance of a computer controlled opponent in a game may be encountered by many different human players each using different strategies to try to defeat it. The possibilities exist here that distributed training through possibly millions of encounters with online players, an AIS strategy collection may develop to the point where it lends itself to the overall persona of the game characters that are associated with it.

**Further work**

With regards to the AIS implementation in the Robocode environment, the granularity of the tactics was quite large, insofar as that each tactic in itself could be considered a strategy. In other adaptive and learning environments, the components of the strategies have been much more basic. Eisenstein (Eisenstein, 2003) showed that evolving Robocode robots could be achieved with much finer grained tactics (Figure 9).
Figure 9 A sample Robocode program used to control a Robocode robot from Eisenstein’s TableRex program (Eisenstein, 2003)

<table>
<thead>
<tr>
<th>Function</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Random</td>
<td>Ignore</td>
<td>Ignore</td>
<td>0.87</td>
</tr>
<tr>
<td>2. Divide</td>
<td>Const_1</td>
<td>Const_2</td>
<td>0.5</td>
</tr>
<tr>
<td>3. Greater Than</td>
<td>Line 1</td>
<td>Line 2</td>
<td>1</td>
</tr>
<tr>
<td>4. Normalize Angle</td>
<td>Enemy bearing</td>
<td>Ignore</td>
<td>-50</td>
</tr>
<tr>
<td>5. Absolute Value</td>
<td>Line 4</td>
<td>Ignore</td>
<td>50</td>
</tr>
<tr>
<td>6. Less Than</td>
<td>Line 4</td>
<td>Const_90</td>
<td>1</td>
</tr>
<tr>
<td>7. Add</td>
<td>Line 6</td>
<td>Line 3</td>
<td>1</td>
</tr>
<tr>
<td>8. Multiply</td>
<td>Const_10</td>
<td>Const_10</td>
<td>100</td>
</tr>
<tr>
<td>9. Less Than</td>
<td>Enemy distance</td>
<td>Line 3</td>
<td>0</td>
</tr>
<tr>
<td>10. Add</td>
<td>Line 9</td>
<td>Line 7</td>
<td>0</td>
</tr>
<tr>
<td>11. Multiply</td>
<td>Line 10</td>
<td>Line 4</td>
<td>0</td>
</tr>
<tr>
<td>12. Output</td>
<td>Turn gun left</td>
<td>Line 11</td>
<td>0</td>
</tr>
</tbody>
</table>

Similarly Hong and Cho (Hong & Cho, 2004) have a complex set of rules which could be augmented to modify a Robocode robots behaviour. What our approach and these and many others have taken is that the strategy is encoded into a chromosome which is a representation of the current strategy the robot is using. In the case of our implementation, the strategy length variable controlled this parameter. It is conceivable that breaking the tactics down into their core components and using these to build the maximum strategies could lead to markedly better results. However, the time it would take to learn or adapt strategies of great length may make such granularity unusable in a game environment.

Another item that was not addressed in this study is the generalization and dilution of strategies as the AIS system learns and adapts. As in the biological immune system, it is the procedure that our acquired immunity, preserved through memory B Cells, is not perpetual. This is also the case in our AIS. If a strategy is not encountered for a long time, the natural process of adapting to later strategies will, in due course, cause the appropriate strategy for that encounter to be subsumed by latter strategies. Now whether this strategy is gone completely, or whether some part of it still remains in later strategies still remains
to be seen, but as almost all strategies are some ancestor of previous generations, it is highly plausible that the latter is the most likely. If this is the case, then there should remain some level of adaptation performance to a certain opponent strategy, even if that strategy has not been encountered for a long time.
Works Cited


### Appendix A
Example of successful strategy generation over ten generations

<table>
<thead>
<tr>
<th>Strategy 1</th>
<th>Probability 1</th>
<th>Strategy 2</th>
<th>Probability 2</th>
<th>Strategy 3</th>
<th>Probability 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEFEEGCF</td>
<td>0.07</td>
<td>HF</td>
<td>0.23</td>
<td>HF</td>
<td>0.33</td>
</tr>
<tr>
<td>FBGJCBG</td>
<td>0.07</td>
<td>BEFEEGCF</td>
<td>0.3</td>
<td>BEFEEGCF</td>
<td>0.45</td>
</tr>
<tr>
<td>A</td>
<td>0.07</td>
<td>FBGJCBG</td>
<td>0.33</td>
<td>FBGJCBG</td>
<td>0.45</td>
</tr>
<tr>
<td>JBFDFJ</td>
<td>0.09</td>
<td>A</td>
<td>0.32</td>
<td>A</td>
<td>0.41</td>
</tr>
<tr>
<td>IGFHDJBCI</td>
<td>0.09</td>
<td>JBFDFJ</td>
<td>0.36</td>
<td>JBFDFJ</td>
<td>0.5</td>
</tr>
<tr>
<td>ADJBHJID</td>
<td>0.07</td>
<td>CJBHIH</td>
<td>0.17</td>
<td>CJBHIH</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGFHDJBCI</td>
<td>0.36</td>
<td>IGFHDJBCI</td>
<td>0.48</td>
</tr>
<tr>
<td>HF</td>
<td>0.09</td>
<td>ADJBHJID</td>
<td>0.14</td>
<td>EGIIBGIGC</td>
<td>0.39</td>
</tr>
<tr>
<td>BEFEEGCF</td>
<td>0.13</td>
<td>EGIIBGIGC</td>
<td>0.32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FBGJCBG</td>
<td>0.14</td>
<td>HF</td>
<td>0.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0.15</td>
<td>HF</td>
<td>0.24</td>
<td>BEFEEGCF</td>
<td>0.49</td>
</tr>
<tr>
<td>JBFDFJ</td>
<td>0.16</td>
<td>BEFEEGCF</td>
<td>0.36</td>
<td>FBGJCBG</td>
<td>0.49</td>
</tr>
<tr>
<td>CJBHIH</td>
<td>0.07</td>
<td>FBGJCBG</td>
<td>0.37</td>
<td>A</td>
<td>0.46</td>
</tr>
<tr>
<td>IGFHDJBCI</td>
<td>0.17</td>
<td>A</td>
<td>0.37</td>
<td>JBFDFJ</td>
<td>0.54</td>
</tr>
<tr>
<td>ADJBHJID</td>
<td>0.14</td>
<td>JBFDFJ</td>
<td>0.4</td>
<td>CJBHIH</td>
<td>0.29</td>
</tr>
<tr>
<td>EGIIBGIGC</td>
<td>0.09</td>
<td>CJBHIH</td>
<td>0.17</td>
<td>IGFHDJBCI</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGFHDJBCI</td>
<td>0.4</td>
<td>EGIIBGIGC</td>
<td>0.43</td>
</tr>
<tr>
<td>HF</td>
<td>0.17</td>
<td>ADJBHJID</td>
<td>0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEFEEGCF</td>
<td>0.19</td>
<td>EGIIBGIGC</td>
<td>0.37</td>
<td>HF</td>
<td>0.42</td>
</tr>
<tr>
<td>FBGJCBG</td>
<td>0.21</td>
<td></td>
<td></td>
<td>BEFEEGCF</td>
<td>0.51</td>
</tr>
<tr>
<td>A</td>
<td>0.2</td>
<td>HF</td>
<td>0.29</td>
<td>FBGJCBG</td>
<td>0.52</td>
</tr>
<tr>
<td>JBFDFJ</td>
<td>0.24</td>
<td>BEFEEGCF</td>
<td>0.41</td>
<td>A</td>
<td>0.51</td>
</tr>
<tr>
<td>CJBHIH</td>
<td>0.15</td>
<td>FBGJCBG</td>
<td>0.41</td>
<td>JBFDFJ</td>
<td>0.58</td>
</tr>
<tr>
<td>IGFHDJBCI</td>
<td>0.24</td>
<td>A</td>
<td>0.41</td>
<td>CJBHIH</td>
<td>0.33</td>
</tr>
<tr>
<td>ADJBHJID</td>
<td>0.17</td>
<td>JBFDFJ</td>
<td>0.44</td>
<td>IGFHDJBCI</td>
<td>0.54</td>
</tr>
<tr>
<td>EGIIBGIGC</td>
<td>0.17</td>
<td>CJBHIH</td>
<td>0.24</td>
<td>EGIIBGIGC</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IGFHDJBCI</td>
<td>0.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HF</td>
<td>0.23</td>
<td>EGIIBGIGC</td>
<td>0.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEFEEGCF</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FBGJCBG</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JBFDFJ</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CJBHIH</td>
<td>0.17</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IGFHDJBCI</td>
<td>0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADJBHJID</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EGIIBGIGC</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Appendix B**

A list of commands available to the Robocode robot

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ahead</td>
<td>Moves your robot forward.</td>
</tr>
<tr>
<td>back</td>
<td>Moves your robot backward.</td>
</tr>
<tr>
<td>doNothing</td>
<td>Do nothing this turn.</td>
</tr>
<tr>
<td>finalize</td>
<td>Called by the system to 'clean up' after your robot.</td>
</tr>
<tr>
<td>fire</td>
<td>Fires a bullet.</td>
</tr>
<tr>
<td>fireBullet</td>
<td>Fires a bullet.</td>
</tr>
<tr>
<td>getBattleFieldHeight</td>
<td>Get height of the current battlefield.</td>
</tr>
<tr>
<td>getBattleFieldWidth</td>
<td>Get width of the current battlefield.</td>
</tr>
<tr>
<td>getEnergy</td>
<td>Returns the robot's current energy</td>
</tr>
<tr>
<td>getGunCoolingRate</td>
<td>Returns the rate at which the gun will cool down.</td>
</tr>
<tr>
<td>getGunHeading</td>
<td>Returns gun heading in degrees.</td>
</tr>
<tr>
<td>getGunHeat</td>
<td>Returns the current heat of the gun.</td>
</tr>
<tr>
<td>getHeading</td>
<td>Returns the direction the robot is facing, in degrees.</td>
</tr>
<tr>
<td>getHeight</td>
<td>Returns the height of the robot</td>
</tr>
<tr>
<td>getName</td>
<td>Returns the robot's name</td>
</tr>
<tr>
<td>getNumRounds</td>
<td>Returns the number of rounds in the current battle</td>
</tr>
<tr>
<td>getOthers</td>
<td>Returns how many opponents are left</td>
</tr>
<tr>
<td>getRadarHeading</td>
<td>Returns radar heading in degrees.</td>
</tr>
<tr>
<td>getRoundNum</td>
<td>Returns the number of the current round (1 to getNumRounds()) in the battle</td>
</tr>
<tr>
<td>getTime</td>
<td>Returns the current game time Note: 1 battle consists of multiple rounds Time is reset to 0 at the beginning of every round.</td>
</tr>
<tr>
<td>getVelocity</td>
<td>Returns the velocity of the robot.</td>
</tr>
<tr>
<td>getWidth</td>
<td>Returns the width of the robot</td>
</tr>
<tr>
<td>getX</td>
<td>Returns the X position of the robot.</td>
</tr>
<tr>
<td>getY</td>
<td>Returns the Y position of the robot.</td>
</tr>
<tr>
<td>onBulletHit</td>
<td>This method will be called when one of your bullets hits another robot.</td>
</tr>
<tr>
<td>onBulletHitBullet</td>
<td>This method will be called when one of your bullets hits another bullet.</td>
</tr>
<tr>
<td>onBulletMissed</td>
<td>This method will be called when one of your bullets misses (hits a wall).</td>
</tr>
<tr>
<td>onDeath</td>
<td>This method will be called if your robot dies You should override it in your robot if you want to be informed of this event/</td>
</tr>
<tr>
<td>onHitByBullet</td>
<td>This method will be called when your robot is hit by a bullet.</td>
</tr>
<tr>
<td>onHitRobot</td>
<td>This method will be called when your robot collides</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onHitWall</td>
<td>This method will be called when your robot collides with a wall.</td>
</tr>
<tr>
<td>onRobotDeath</td>
<td>This method will be called if another robot dies. You should override it in your robot if you want to be informed of this event.</td>
</tr>
<tr>
<td>onScannedRobot</td>
<td>This method will be called when your robot sees another robot.</td>
</tr>
<tr>
<td>onWin</td>
<td>This method will be called if your robot wins a battle. Actions will have no effect if called from this section.</td>
</tr>
<tr>
<td>resume</td>
<td>Resume the movement you stopped in stop(), if any.</td>
</tr>
<tr>
<td>run</td>
<td>The main method in every robot.</td>
</tr>
<tr>
<td>scan</td>
<td>Look for other robots.</td>
</tr>
<tr>
<td>setAdjustGunForRobotTurn</td>
<td>Sets the gun to automatically turn the opposite way when the robot turns.</td>
</tr>
<tr>
<td>setAdjustRadarForGunTurn</td>
<td>Sets the radar to automatically turn the opposite way when the gun turns.</td>
</tr>
<tr>
<td>setAdjustRadarForRobotTurn</td>
<td>Sets the radar to automatically turn the opposite way when the robot turns.</td>
</tr>
<tr>
<td>setColors</td>
<td>Call this method to set your robot's colors.</td>
</tr>
<tr>
<td>setRobotPeer</td>
<td>This method is called by the game.</td>
</tr>
<tr>
<td>stop</td>
<td>Stops all movement, and saves it for a call to resume().</td>
</tr>
<tr>
<td>turnGunLeft</td>
<td>Rotates your robot's gun.</td>
</tr>
<tr>
<td>turnGunRight</td>
<td>Rotates your robot's gun.</td>
</tr>
<tr>
<td>turnLeft</td>
<td>Rotates your robot.</td>
</tr>
<tr>
<td>turnRadarLeft</td>
<td>Rotates your robot's radar.</td>
</tr>
<tr>
<td>turnRadarRight</td>
<td>Rotates your robot's radar.</td>
</tr>
<tr>
<td>turnRight</td>
<td>Rotates your robot.</td>
</tr>
</tbody>
</table>