Character Interaction Simulation Engine

Shaun Gray
B.A. (Mod.) CS
Final Year Project, April 2011
Supervisor: Dr. Stefan Weber
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

______________________________  April 11, 2011
Shaun Gray
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

________________________________________ April 11, 2011
Shaun Gray
Acknowledgements

I would like to thank Dr. Stefan Weber for his advice and guidance throughout the course of this project. I would also like to thank my family and friends for their continuous support throughout my college career; without this support I would not be here today.

A great deal of gratitude is also owed to Emer Wall for her support and guidance with the genetic, evolutionary and biological aspects of this project.

Finally, I would like to thank my sister Ruth, and good friend Karen Forde for proof-reading many drafts of this thesis.
‘.. every man is the architect of his own fortune’
– Gaius Sallustius Crispus
# Contents

**Preamble** ............................................. i  
Declaration ........................................... i  
Permission to Lend .................................... ii  
Acknowledgements ....................................... iii  

**1 Introduction** ....................................... 1  
1.1 Motivation ......................................... 2  
1.2 Objectives ......................................... 3  
1.3 Report Layout ...................................... 3  
1.4 Summary ............................................ 5  

**2 State of the Art** ..................................... 6  
2.1 Evolutionary Game Theory ............................ 7  
2.2 Cooperation and The Prisoner’s Dilemma ............. 8  
2.3 Altruism ............................................ 9  
2.4 Simulation Engines ................................... 11  
  2.4.1 RAM ............................................. 11  
  2.4.2 The Genesys/Tracker System ..................... 11  
  2.4.3 Generic Biological Simulator .................... 12  
  2.4.4 Discussion of Related Simulation Engines ...... 13  
2.5 Simulation Games .................................... 14  
  2.5.1 Creatures ....................................... 14  
  2.5.2 Tamagotchi ...................................... 14  
  2.5.3 Spore ............................................ 15  
  2.5.4 The Sims ........................................ 16  
  2.5.5 Discussion of Related Games .................... 17  
2.6 Summary ............................................ 18  

**3 Design** .............................................. 19  
3.1 Discussion of the Problem Domain .................. 20  
  3.1.1 Aims ............................................ 20
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1.2 Factors Which Govern Interaction</td>
<td>21</td>
</tr>
<tr>
<td>3.2 Characters</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Communities</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Environment</td>
<td>28</td>
</tr>
<tr>
<td>3.5 Simulation</td>
<td>29</td>
</tr>
<tr>
<td>3.6 Aims</td>
<td>29</td>
</tr>
<tr>
<td>3.7 Rules</td>
<td>30</td>
</tr>
<tr>
<td>3.8 XML - Representing Components</td>
<td>31</td>
</tr>
<tr>
<td>3.9 Displaying Results</td>
<td>34</td>
</tr>
<tr>
<td>3.10 Class Architecture</td>
<td>35</td>
</tr>
<tr>
<td>3.11 Engine Algorithm</td>
<td>37</td>
</tr>
<tr>
<td>3.12 Movement &amp; Pathing Algorithms</td>
<td>40</td>
</tr>
<tr>
<td>3.13 Loops Versus Threads</td>
<td>42</td>
</tr>
<tr>
<td>3.14 Development Model</td>
<td>43</td>
</tr>
<tr>
<td>3.15 Summary</td>
<td>43</td>
</tr>
<tr>
<td>4 Implementation</td>
<td>44</td>
</tr>
<tr>
<td>4.1 Language Chosen</td>
<td>45</td>
</tr>
<tr>
<td>4.2 Development Environment</td>
<td>45</td>
</tr>
<tr>
<td>4.2.1 Platform</td>
<td>45</td>
</tr>
<tr>
<td>4.2.2 Eclipse</td>
<td>46</td>
</tr>
<tr>
<td>4.2.3 Backup and Versioning</td>
<td>46</td>
</tr>
<tr>
<td>4.3 Logging</td>
<td>47</td>
</tr>
<tr>
<td>4.4 XML Parsing and Writing</td>
<td>48</td>
</tr>
<tr>
<td>4.5 Summary</td>
<td>48</td>
</tr>
<tr>
<td>5 Evaluation</td>
<td>49</td>
</tr>
<tr>
<td>5.1 The Finished Product</td>
<td>50</td>
</tr>
<tr>
<td>5.1.1 Successes</td>
<td>50</td>
</tr>
<tr>
<td>5.1.2 Limitations</td>
<td>50</td>
</tr>
<tr>
<td>5.2 Testing</td>
<td>51</td>
</tr>
<tr>
<td>5.2.1 Interaction</td>
<td>51</td>
</tr>
<tr>
<td>5.2.2 Scalability</td>
<td>56</td>
</tr>
<tr>
<td>5.3 Summary</td>
<td>56</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>57</td>
</tr>
<tr>
<td>6.1 Discussion of the Project</td>
<td>58</td>
</tr>
<tr>
<td>6.2 Future Work</td>
<td>58</td>
</tr>
<tr>
<td>6.2.1 Environmental Interaction</td>
<td>58</td>
</tr>
<tr>
<td>6.2.2 Reproduction and Evolution Algorithms</td>
<td>59</td>
</tr>
<tr>
<td>6.2.3 Attack Algorithms</td>
<td>60</td>
</tr>
</tbody>
</table>
CONTENTS

6.3 Reflection .............................................. 60
6.4 Summary .............................................. 61

Bibliography .............................................. 62
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Prisoner’s Dilemma Payoff Matrix</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>‘Ants’ evolving to traverse a winding path</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>A ‘Norn’ character, from Creatures</td>
<td>14</td>
</tr>
<tr>
<td>2.4</td>
<td>A Tamagotchi Connection V1</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Creating a Character in Spore</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>Tree of Life (ToL)</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>10x10 Environment</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>XML representation of a Character</td>
<td>32</td>
</tr>
<tr>
<td>3.4</td>
<td>XML representation of rules and aims</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>XML representation of a Community</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>XML representation of an Environment</td>
<td>34</td>
</tr>
<tr>
<td>3.7</td>
<td>XML representation of a Simulation</td>
<td>34</td>
</tr>
<tr>
<td>3.8</td>
<td>Snippet of a sample XML log file</td>
<td>35</td>
</tr>
<tr>
<td>3.9</td>
<td>UML Class Diagram for the Simulation Engine</td>
<td>36</td>
</tr>
<tr>
<td>3.10</td>
<td>Simulation Engine Flowchart</td>
<td>39</td>
</tr>
<tr>
<td>3.11</td>
<td>Moving to a point</td>
<td>40</td>
</tr>
<tr>
<td>3.12</td>
<td>Characters as constraints</td>
<td>41</td>
</tr>
<tr>
<td>5.1</td>
<td>Percentage Strength Vs. Age in Humans</td>
<td>52</td>
</tr>
<tr>
<td>5.2</td>
<td>Percentage Strength Vs. Age in Human Characters</td>
<td>53</td>
</tr>
<tr>
<td>5.3</td>
<td>Characters Moving Towards each other to Mate</td>
<td>54</td>
</tr>
<tr>
<td>5.4</td>
<td>A ‘Cat and Mouse’ game between two Characters</td>
<td>55</td>
</tr>
<tr>
<td>6.1</td>
<td>One-point Crossover (Riccardo Poli and W.B. Langdon, 1997)</td>
<td>60</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Comparison of Simulation Engines .......................... 13
2.2 Comparison of Simulation Games ......................... 17
3.1 Visual representation of an aim list ....................... 30
3.2 Sample constraint set for ‘Reproduction’ and ‘Perceived Threat’ rules ........................................ 31
5.1 Running Times ............................................... 56
Abstract

Simulation of character interaction can be employed to model the interactions of characters and communities over time and provides indications of the development and evolution of populations. The simulations are based on simulation engines that interpret the development of attributes of individual characters in response to the presence of other characters and to the environment.

Most simulation engines in this domain, however, suffer from either being very specific to their problem domain or do not allow the specification of realistic, real-world attributes of characters. A simulation engine that addresses these issues requires a vast amount of parameters in order to deliver a realistic simulation. It needs to take into account factors which govern interaction between characters; their aims, memory, threat perception, ability to reproduce, altruism, etc, as well as as a defined set of rules, goals, and character-specific attributes. The specification of these attributes and the implementation of an appropriate simulation engine requires an understanding of the underlying science of nature in order to define realistic behaviour.

In my project, I investigated the development of a simulation engine for character interaction that allows the specification of realistic attributes for characters and the formation of communities from these characters. The implementation demonstrates that such a simulation engine can be used to evaluate realistic scenarios and predict the outcome of changes of attributes of individual characters and communities.
Chapter 1

Introduction
In this chapter an introduction to the problem domain for this project will be given. Initially the motivation behind this project will be discussed, followed by a set of objectives that this project aims to accomplish. Finally there will be a brief summarisation of this report’s layout and a summary of the topics that this chapter covers.

1.1 Motivation

The initial motivation behind this project arose after attempting to answer the question as to who would win in a fight: Pirates or Ninjas? The Pirates versus Ninjas debate, or PvN, is quite a common debate nowadays and has become something of an internet meme 1.

Although the PvN seems to be quite a humorous and frivolous topic at the surface, I hypothesise that upon close examination and by breaking the problem down to its most basic form, it becomes a question of aim-based character interaction. In essence, the PvN is asking what is the outcome of having two communities of characters interact, where characters from each community have their own aims and interact and move based on these aims and basic rules. (The aims in this case being to kill Pirates or Ninjas.)

In an attempt to answer this question I began research into all of the variables involved in such interaction. Each character has their own motivation for interaction, their own goals so to speak, and each character also has their own attributes. As these were Pirates and Ninjas, characters from the real world and not entirely fictitious creations, each character’s attributes would also be realistic, or at the very least indicative of real world values.

While researching I also discovered several games and books based on the PvN. For example, ‘Pirates vs. Ninjas Dodgeball’ for the Xbox and Wii games consoles. This game allowed play between various groups, not just Pirates and Ninjas but also incorporating Zombies and Robots. While Zombies are fictional creatures, this still led me to consider a more generic simulator. Instead of simply simulating the interaction between Pirates and Ninjas, it would be interesting if an engine was written which would allow a user to define their own characters and groups of characters, and furthermore model their interaction. It would be even more interesting, and indeed beneficial, if the engine allowed for the use of real world data so that not only could

\[1\text{A meme is a concept, catchphrase or byword that spreads from person to person (mem, 2003)}\]
CHAPTER 1. INTRODUCTION

humorous topics like the PvN be answered but also to allow for the modelling of actual animal and organism interaction over time.

In effect, the PvN had compelled me to write a generic, aim-based interaction simulation engine, in which a user could define their own characters and communities and allow them to use real world data in doing so.

1.2 Objectives

This project aspires to create a simulation engine which can be used to create characters and communities with the intention of viewing the outcome of realistic aim-based community and character interaction over a period of time.

The key points to note are as follows:

- The engine must allow characters to be defined, and must allow real world data to be used for character attributes.
- The definition of a character may be derived from a generic template, i.e. a set of characters can be created from the same template.
- The engine must be capable of modelling community and character interaction, in real-time or otherwise.
- The results of character interactions should be clearly and succinctly displayed.
- Character interactions must be able to defined through simple rule sets and these rules must be easily modified.
- Characters must evolve over time.
- Characters in the engine must be able to reproduce.
- The engine must allow communities to be defined.
- The engine must allow the definition of environments. Though environment-character interaction is not being modelled.

1.3 Report Layout

The remainder of this report is laid out as follows:
CHAPTE41. INTRODUCTION

Chapter 2: This chapter will provide a review of publications and other works related to this project, which have had a direct influence on its design and implementation. First, it will discuss publications on the topics of animal behaviour, interaction, evolution, altruism and reproduction. This will then be followed by an examination of other simulation engines. Finally, there will be an overview of applications that address issues similar to the ones in this project.

Chapter 3: This chapter outlines what was considered for the implementation of the application created as part of this project. Initially, there will be a discussion of the problem domain, namely the factors that govern character interaction. This will be followed by a breakdown of all of the components that went into this project and a discussion of how each component was designed and the reason(s) behind that design. There will then be an overview of the class architecture and engine and movement algorithms and a discussion about why loops were favoured over threads. Finally the development model employed in this project will be discussed.

Chapter 4: In this chapter, an overview of the implementation of the simulation engine is given. To begin, the choice of programming language used is discussed. This is followed by a discussion of the development environment. Finally some language specific implementations - logging and XML parsing/writing - are discussed.

Chapter 5: This chapter views the completed project and evaluates it against the aims which were laid out at the beginning. First of all it looks at the project in its entirety and what elements were completed and which were not. It then looks at several tests which were performed to judge how well the project was implemented.

Chapter 6: This chapter will discuss the conclusions that were reached during and after the development of this simulation engine. First there will be an overview of the project as a whole. This will be followed by a discussion of future work - features which can be added and directions in which the project can go. Finally it will consider everything that was learned throughout the course of the project.
1.4 Summary

In this chapter the initial motivation behind this project was discussed. This was followed by a set of objectives for the project. Finally, a brief overview of the rest of this report was given.
Chapter 2

State of the Art
CHAPTER 2. STATE OF THE ART

This chapter will provide a review of publications and other works related to this project, which have had a direct influence on its design and implementation. First, it will discuss publications on the topics of animal behaviour, interaction, evolution, altruism and reproduction. This will then be followed by an examination of other simulation engines. Finally, there will be an overview of applications that address issues similar to the ones in this project.

2.1 Evolutionary Game Theory

Games Theory was originally developed by John Von Neumann and Oskar Morgenstern (1944). It is a branch of applied mathematics that attempts to capture behaviour in strategic situations. The most important contribution that it has made to evolutionary biology is the concept of Evolutionary Stable Strategies (ESS). This concept is central to modern evolutionary ecology and Dawkins (1989) has stated that it could be ‘one of the most important advances in evolutionary theory since Darwin’. This concept was originally brought into ecology by John Maynard Smith and George R. Price (1973) and the field of evolutionary game theory is now well-established in ecology, biology and economics.

According to Smith (1982), “An ‘ESS’, or ‘evolutionary stable strategy’ is a strategy such that, if all the members of a population adopt it, no mutant strategy can invade”. There may be more than one ESS for a population and there are many determining factors for the type(s) of ESS. Such factors include: the characteristics of the members of a population, population size, whether population members can learn from past encounters, how members of a population reproduce and so on. Dawkins (1989) contains a discussion of ESS with many clearly explained examples.

A very simple, theoretical example of an ESS is as follows (as discussed by Smith and Dawkins): Assume only two strategies are possible, i.e. hawk (fight as hard as is possible, retreat when badly hurt) and dove (threaten but never hurt the opponent). Assume also that individuals do not learn from previous experience i.e. they cannot tell if an individual is a hawk or dove upon encountering them. In a population made up of all doves, all doves do well. However, introduction of a single ‘mutant’ hawk will put all doves to...
at a disadvantage as it always beats them and the hawk gene will begin to spread throughout the environment. However, now that hawks increase in frequency, the hawks will encounter each other and get hurt. Consequently, even a single dove is then at an advantage because it will not get hurt and the dove gene will increase and become more common. A stable ratio of hawks/doves is reached when the average payoff is equal for both.

Although this field is primarily attributed to John Maynard Smith, W.D. Hamilton ‘played an important, and indeed pioneering role’ (Sigmund, 2002). Hamilton, a ‘revolutionary’, evolutionary biologist and theorist published extensively on the topic of altruism and kin selection. Hamilton published extensively on the topic of altruism and kin selection. In fact, when Smith first introduced the definition of the ESS he stressed that it had ‘been derived in part from the theory of games, and in part from the work of MacArthur and Hamilton’ (Maynard Smith and Price, 1973).

### 2.2 Cooperation and The Prisoner’s Dilemma

Axelrod and Hamilton (1981) contains an interesting discussion on both the Prisoner’s Dilemma and cooperation between creatures.

The Prisoner’s Dilemma - as elaborated by Redmond (2008, pp. 9–13) - is a fundamental problem in game theory which demonstrates why two individuals may not cooperate, even if it is in their best interests to do so. Figure 2.1 shows the payoff matrix of the Prisoner’s Dilemma. From this figure it can be seen that the ‘best’ solution to the Prisoner’s Dilemma game is to always defect, if there is no chance of encountering the other character again. This can be determined as follows: if one player cooperates and the other does not the payoff is $T$ (the temptation to defect). If, however, the other cooperates the payoff is $R$ (the reward for mutual cooperation). By assumption, $T > R$ so it pays to defect if the other cooperates. However, if the other player defects, there is a choice between either cooperation, which yields $S$ (the sucker’s payoff) or defection, which yields $P$ (the punishment for mutual defection). Again, by assumption $P > S$ so it pays to defect if the other defects. Therefore, it always pays to defect, even though if both defect they receive $P$, and not the larger value of $R$. Axelrod and Hamilton (1981) remark that with ‘two individuals destined never to meet again, the

---

2Kin selection is an evolutionary theory that suggests that individuals are more likely to help blood relatives, to increase the odds of gene transmission.
only strategy that can be called a solution to the game is to defect always de-
spite the seemingly paradoxical outcome that both do worse than they could
have had they cooperated'. Applying this theory to biological evolution, a
similar situation arises. If the payoffs are in terms of fitness and there is
random interaction between characters (which are not repeated), then 'any
population with a mixture of heritable strategies evolves to a state where all
individuals are defectors' (idem).

\[ \begin{array}{c|c|c|c}
 & C & D \\
\hline
C & R=3 & S=0 \\
\hline
D & T=0 & P=1 \\
\end{array} \]

Figure 2.1: Prisoner’s Dilemma Payoff Matrix

It is not uncommon for individuals to meet more than once in many biological
settings. If an individual is capable of remembering previous engagements
with other individuals, or some aspect(s) of the outcome of prior engage-
ments, then the strategic situation mentioned above becomes an iterated
Prisoner’s Dilemma. This would have the affect that a strategy would now
take the form of a ‘decision rule’ which would determine the probability of de-
fection or cooperation as a ‘function of the history of the interaction’ (idem)
so far.

2.3 Altruism

Ridley (1995, pp. 234) defines altruism as ‘the transfer of some benefit from
the altruist [a character] to the recipient, at a cost to the altruist’.

Altruism is interesting in biology in that it seems to contradict the the-
ory of Natural Selection \(^3\). Altruistic traits, by their very definition, have a

\(^3\)Darwin (1958) coined the term Natural Selection, in reference to evolution – “I have
called this principle, by which each slight variation, if useful, is preserved, by the term
Natural Selection”
detrimental effect on reproduction levels. An altruistic act may be defined in terms of offspring, given that offspring can be used as a means of measuring benefits and costs. An altruistic act is ‘one that increases the reproduction of the recipient and decreases that of the altruist’ (idem). This seems contradictory and yet it is not. Altruistic behaviour which fits this definition does indeed exist in nature and can be observed in ant colonies or bee hives and other sterile worker castes of social insects. These castes work to increase the reproductive capabilities of another and do not reproduce themselves.

Axelrod and Hamilton (1981) provides an interesting discussion on altruism and the effects of altruistic personalities on cooperation between species.

‘If interactants are sufficiently closely related, altruism can benefit reproduction of the set, despite losses to the individual altruist. In accord with this theory’s predictions, apart from the human species, almost all clear cases of altruism, and most observed cooperation, occur in contexts of high relatedness, usually between immediate family members.’

A clear example of this form of altruism in closely related family members can be seen in the Malaysian seed-gathering ant 4. When this ant is attacked, or defending the colony, it contracts its abdominal muscles with sufficient strength to split its cuticle. The sticky contents of the mandibular gland then burst out and the glue immobilizes the opponent. The ant’s self-sacrifice should benefit its colony (more information can be found in Ridley (1995, pp. 1–4)).

Despite the aforementioned, Axelrod and Hamilton also describe situations where altruistic cooperative behaviour is exhibited in species ‘where relatedness is low or absent’, stating that ‘cooperation is common between members of the same species and even members of different species’. Examples given include the fig wasp and fig tree, ‘where wasps, which are obligate parasites of fig flowers, serve as the tree’s sole means of pollination and seed set’.

Further reading on altruism and altruistic behaviour can be found in Ridley (1995, pp. 234 – 251) and a broader overview in Dawkins (1989).

---

4A close relation of the species Camponotus saundersi, which was studied in detail by Ulrich and Eleanore Maschwitz
2.4 Simulation Engines

Schwetman (1996) describes a simulation engine as ‘the collection of components, features and support functions which are crucial to the implementation of an efficient discrete event simulation model’. Robinson (2004, pp. 2) provides a more comprehensive definition, defining a computer based simulation which acts over time as ‘an imitation (on a computer) of a system as it progresses through time.’

There are currently a number of simulation engines available for modelling changes in biological and social situations and for modelling natural evolution. However, these often take the form of fine-grained simulators which are too specific to one particular field, or simulators where populations are represented as programs. In the following, several such engines will be discussed.

2.4.1 RAM

RAM is an example of an early approach to evolutionary simulation and the simulation of character interaction. Taylor et al. (1989) report on this simulator, in which each organism is represented as a pair containing both a parametrized Lisp function and a sequence of parameter values to the behaviour function. In RAM, characters produce asexually and exist in a common environment, in which they interact. RAM is however a very limited system. Langton (1997, pp. 5) describes the systems limitations in detail.

‘The “genes” defined a relatively small parameter space within which variation could occur, leaving limited scope for innovation and evolution ... at the time it was built only a few hundred individuals could be simulated for a few tens of generations per hour of workstation time ...’

2.4.2 The Genesys/Tracker System

Jefferson et al. (2004) drastically extended the idea of representing organisms as programs with systems such as Genesys. In this system, organisms are represented as a computer system whose ‘execution represents the sequence of significant events in the organism’s life’. Essentially, each organism is either a simulated neural net or a finite state automata and its genes are represented as bit strings. Langton (1997) and Jefferson et al. (2004) discuss

5 Though paired reproduction can be specially programmed in.
how, over the course of several generations, this simulation engine is capable
of modelling the evolution of individuals’ abilities to traverse a winding path
in a rectilinear grid. To do this, an ‘ant’ is placed in a two-dimensional grid
at the beginning of the broken trial, as shown in figure 2.2. This grid wraps
around so that cells on the left edge are adjacent to those on the right and
cells at the top edge are adjacent to those at the bottom. The ant then moves
from cell to cell. The ant’s success is measured by the number of cells on the
trail that it manages to traverse. An ant is only able to see the cells directly
ahead of it and only able to perform one action at a time: move forward
one step, turn left/right or no-op. In order for the ants to evolve to traverse
the trail they must embody an algorithm which causes them to move forward
when they see a trail cell ahead, or to search locally if one cannot be seen.

![Figure 2.2: ‘Ants’ evolving to traverse a winding path](image)

2.4.3 Generic Biological Simulator

Milazzo (2005) describes in detail the Generic Biological Simulator, or GBS.
The GBS is a biological simulator, written in Java, which can be specialized
for various fields within biology. As Milazzo states, ‘the specialization to a
particular field can be achieved by implementing some Java interfaces, pro-
vided by the tool, and by making the tool aware of the new implementations’,
however the engine itself never changes. The main idea behind the engine
is that systems being simulated are represented as multisets that evolve by
‘means of applications of rewriting rules’.
2.4.4 Discussion of Related Simulation Engines

Each of the engines discussed bears some resemblance to this project. Each simulator allows for the evolution of organisms over time. However, with the exception of GBS, none of the engines allow for the definition of rules. As well as this, none of these simulators allow for the definition of aims to model interaction and evolution or for the use of real world data. Table 2.1 shows a comparison of the engines in terms of what they allow to be defined and modelled.

<table>
<thead>
<tr>
<th></th>
<th>RAM</th>
<th>Genesys/Tracker</th>
<th>GBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Character Template</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Real World Data</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Definition of Communities</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Definition of Environments</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Character Interaction</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Community Interaction</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Definition of rule sets</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Definition of various aims</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Reproduction</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Evolution</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison of Simulation Engines

The GBS, being a generic simulator and allowing for the definition of rule sets, resonates with this project’s goals of allowing a user to create many characters from the same template and allowing rule sets to be defined.
CHAPTER 2. STATE OF THE ART

2.5 Simulation Games

There are a number of computer games which act as simulations in some form or another. In the following, a representative sample of these will be reviewed.

2.5.1 Creatures

Creatures is an artificial life computer game, created in the mid 90’s by Steve Grand. The original Creatures game was set in Albia - a fictional world. There was no set objective and no restriction placed on gameplay. Instead the gameplay was driven by detailed biological and neurological simulation, and the unexpected results they produce. Gameplay was primarily focused on raising alien characters known as Norns, teaching them how to survive, helping them explore, defending them from other characters and breeding them.

![Norn character from Creatures](image)

Figure 2.3: A ‘Norn’ character, from Creatures

Although Creatures utilises neurological and biological simulations, it does not allow for the use of real-world data and instead applied these principles to fictitious creatures.

2.5.2 Tamagotchi

The Tamagotchi is a simple game concept. It is a hand-held digital pet which a user must look after as the pet ages and evolves. The Tamagotchi evolves through various stages of its life, with changes at each stage being based on the pet’s gender, its current generation and the actions that the user has taken thus far. There are a very limited number of actions that a user can

---

6Steve Grand is a computer scientist and internationally recognised roboticist (bbc, 2006)
perform to affect the creature, such as feeding, cleaning, playing games, healing and punishment or praise. If left uncared for, the Tamagotchi will die. Tamagotchi creatures do not interact with each other and as such do not compete in any way. Their evolution and learning is therefore not affected by social interaction.

![Figure 2.4: A Tamagotchi Connection V1](image)

It was estimated in 2006 that over twenty million Tamagotchi units had been sold worldwide (Bandai Inc., 2006), indicating a clear interest in games/applications where the evolution and creation of a character is determined by the user.

### 2.5.3 Spore

Spore is a game from game developer Will Wright\(^7\) which allows a user to control the development of a creature. This creature must compete and survive in a number of environments, each populated with other creatures. When a user has created a character, this character can then be distributed to other players of the game, allowing the initial creator to see how well their creation is doing with other characters.

\(^7\)Wright is co-founder of Maxis, a game development company (now part of Electronic Arts) and is credited with designing The Sims.
There are a number of similarities between Spore and this project. Most notably:

- Users are able to specify how their creation will interact with other creatures.
- A creature’s survival depends not only on its own attributes but also on interaction with other creatures.
- Creatures evolve based on their social interactions with other creatures.

However, the way in which creature’s evolve is only loosely based on actual biological and evolutionary concepts and the game does not allow for evolution and interaction based on real world data.

### 2.5.4 The Sims

The Sims is a video game series which was developed by Maxis and published by Electronic Arts. The games in The Sims series all lack any defined goals. Instead, users can create human characters and place these characters in environments representative of human social settings, such as houses, neighbourhoods and towns. The player is then encouraged to direct their Sims to interact with the environment and other Sims in order to satisfy needs and personal aims or goals. As of April 2008, The Sims franchise had sold over one million copies worldwide, a clear indication of the popularity of (social) interaction simulations where users have fine-grained control over characters (Electronic Arts, 2008).
In the original Sims there are 3 life stages: infant, child and adult. While infants can grow into children, children and adults do not age. Instead, Sims evolve through a point-based experience system, whereby a Sim’s abilities and attributes will increase as they continue to perform similar actions. For example, fitness can be increased by working out regularly.

There are a number of similarities between The Sims and this project. These include:

- Users are able to assign goals or aims for a Sim.
- The way in which Sims operate is affected by interaction with other Sims.
- It is possible to create family units in The Sims, which could be seen as a community of blood-relatives.

### 2.5.5 Discussion of Related Games

There are currently a number of simulation games which are based on the themes of biology, social interaction and evolution. However, these games do not allow for the use of realistic data or provide enough in the way of interaction definition. Table 2.2 shows a comparison of some of these games.

<table>
<thead>
<tr>
<th></th>
<th>Creatures</th>
<th>Tamagotchi</th>
<th>Spore</th>
<th>The Sims</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Character Template</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Real World Data</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Definition of Communities</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Definition of Environments</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Character Interaction</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Community Interaction</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Definition of rule sets</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Definition of various aims</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Reproduction</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Evolution</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.2: Comparison of Simulation Games
As can be seen from this table Spore and The Sims share many similarities with this project, however neither of these satisfy all of this project’s objectives.

2.6 Summary

This chapter reviewed publications and technologies which have had a direct influence on the design and implementation of this project. Initially, publications on the topics of animal behaviour, interaction, evolution, altruism and reproduction were examined. This was followed by an overview of simulation engines which are related to this project in some fashion. Finally, there was a discussion and examination of simulation style games which correspond in some way to this project.
Chapter 3

Design
This chapter outlines what was considered for the implementation of the application created as part of this project. Initially, there will be a discussion of the problem domain, namely the factors that govern character interaction. This will be followed by a breakdown of all of the components that went into this project and a discussion of how each component was designed and the reason(s) behind that design. There will then be an overview of the class architecture and engine and movement algorithms and a discussion about why loops were favoured over threads. Finally the development model employed in this project will be discussed.

3.1 Discussion of the Problem Domain

In order to successfully design a solution to the problem, it is necessary to look at the problem domain in greater detail. This section will do just that. First, by reiterating the goals of this project (as laid out in section 1.2) and then by looking at what it is that makes creatures move and how various attributes can affect their movement and interaction.

3.1.1 Aims

As with any project or application, a well-rounded and clearly thought out design should result in an implementation which fulfils all of the goals set out.

To recap, these are as follows:

- The engine must allow characters to be defined, and must allow real world data to be used for character attributes.
- The definition of a character may be derived from a generic template, i.e. a set of characters can be created from the same template.
- The engine must be capable of modelling community and character interaction, in real-time or otherwise.
- The results of character interactions should be clearly and succinctly displayed.
- Character interactions must be able to defined through simple rule sets and these rules must be easily modified.
- Characters must evolve over time.
- Characters in the engine must be able to reproduce.
• The engine must allow communities to be defined.
• The engine must allow the definition of environments. Though environment-character interaction is not being modelled.

### 3.1.2 Factors Which Govern Interaction

In order to design an engine centered around interaction, it is necessary to first look at the factors which govern interaction between characters.

There is a multitude of variables that must be considered when dealing with interaction between two or more characters. Many of these were discussed in Chapter 2, such as altruism and inter-species cooperation or defection, however, there are many more which must also be considered. These are as follows:

**Character Aims**

Aims describe what it is that a character wants out of life, either at a conscious or subconscious level. For a more in depth philosophical discussion on animal aims see Savater (2002).

This is perhaps the most important aspect to consider when talking about interaction between characters, as the driving force behind a character has a direct impact on how it will interact with others around it and on how it moves. A character’s aims can also refer to the duties that the character is responsible for, and not just its life goals. These aims can of course change throughout the character’s life. For example, if a female character becomes pregnant, she may change her aims so as to protect herself and her unborn.

It is helpful to imagine a character’s aims as a prioritised list. For each item in the list, if that aim cannot be completed or it is not essential to deal with it at that particular time, move on to the next item. For example, a character may have aims as follows: Protect Offspring, Protect Community Members. In this list, the character will attempt to protect its own offspring before attempting to protect community members. If the character is unable to protect its offspring (because there are not within visible range, not being harmed etc.) then it will attempt to satisfy the next aim on its list, the protection of any community members who are visible and being harmed.
For these reasons, it is logical for each character in the engine to have a prioritised list of aims.

**Memory**

This was discussed briefly in Section 2.2 when talking about the iterated Prisoner’s Dilemma.

If a character is capable of remembering the outcome of interaction with another character this can directly affect interaction the next time they meet. In effect, the probability of acting one way or another becomes directly related to past encounters. This is known as conditional probability, where the chance of some event, $A$, occurring is based on an assumption that some other event, $B$, has occurred. This can be represented as follows:

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(A)}$$

Where $Pr(A|B)$ is the probability of $A$ given $B$, $A$ is the probability of acting in some way and $B$ is the outcome of having encountered the character before.

As memory can play such a large role, it is necessary to allow characters in the engine to have a ‘memory’.

**Perceived threats**

Threats and perceived threats play a large role in character interaction. The way in which characters see others directly affects the way in which they will move and interact. There are many variables in play when it comes to threat perception. Examples of these are physical appearance, pre-determined ill-like (at a genetic level), ‘word-of-mouth’ and past experiences.

Stankowich and Blumstein (2005) contains an interesting discussion and analysis of risk perception in animals. Although they discuss a more prey/predator based relationship, the prey could easily be thought of as threats of any kind; in fact it is quite logical to do so. They state that ‘flight initiation is the distance at which an animal begins to flee from an approaching predator’ and they go on to say that ‘flight initiation distance is an excellent metric with which to quantify an individual’s fearfulness in a particular circumstance’.

For these reasons it is necessary to include an idea of threat perception and ‘flight initiation’ in this engine.
Reproduction

Reproduction plays a very important role in all species. It can be argued that reproduction is in fact the main goal for all species, given that without reproduction species would dwindle into non-existence. Gerrig and Zimbardo (2009, pp. 539) states that ‘according to the evolutionary perspective, the main goal of life is to reproduce so that one can pass on one’s genes’.

There are a huge number of variables to consider when it comes to mating. For example, in human mating characteristics such as height, weight, personality, intelligence values, earlobes and nose breadth can often be taken into account (Buss and Schmitt, 1993). Other variables such as sterility, reproduction range (the period of time when a character is capable of reproducing) and gender must also be taken into account.

Due to the huge impact of reproduction on the way in which organisms move and interact it is necessary to include it in this engine.

Furthermore, for the purposes of this simulation engine, only same-species mating will be considered. (Though there are many theories indicating that inter-species mating is beneficial, such as that proposed by Williamson (2009) and even several documented cases, generally between members of very closely related species, such as the lion/tiger hybrid mentioned in R. I. Pocock (1989)).

Similarly, only mating between characters of opposite genders will be implemented. Although same-sex mating occurs regularly in nature with humans, bonobos, dolphins, ducks and chimpanzees all exhibiting homosexual behaviour it will not be included at this time as ‘little is known about the evolutionary consequences of such behavior’ (Bailey and Zuk, 2009).

Finally there will be no idea of contraception employed in this simulation engine. Instead there will be a percentage chance that each character will produce offspring (this is to mimic conditions such as infertility and impotence) and a percentage chance that each character will produce male offspring (this is due to the fact that the E.S.S for producing offspring is not always 50-50 in all organisms (Dawkins, 1989)).
Altruism

Altruism was discussed at length in section 2.3, though to reiterate, altruism is defined as ‘the transfer of some benefit from the altruist to the recipient, at a cost to the altruist’. Altruism has a direct impact on inter-species co-operation, attack, protection and reproduction.

For these reasons, altruism must be included in this engine.

3.2 Characters

As discussed several times previously, the engine must allow for the definition of characters where real world data can be used for attributes. Further this a generic template must be provided to define characters.

The reason for having a generic template for characters is due to the complexities of life and the large amount of species and sub-species that exist. These complexities can be seen in figure 3.1. In order to grasp a proper idea of how deep the complexities go, the position that human beings occupy within the tree of life has been marked in bold type.

To design a simulation engine capable of incorporating all of the known species of organisms would be far too complicated based on the vast range of variables required. Therefore, the implementation of a generic template provides a more flexible and scalable solution.

In order to allow real-world data to be used in conjunction with a generic template it was necessary to investigate which attributes were common across species. The following is a list of common attributes - some may not seem common at first glance but can be used in an indicative fashion - and brief explanations:

Vision Range

This is the maximum distance that a character is able to see. There is no unit of measurement here as it is a relative value across characters. For example, a human many have a vision range of six whereas a hawk may have a vision range of twenty. Furthermore, all characters are capable of seeing 360°. This is to mimic the ability to turn and look around.
Figure 3.1: Tree of Life (ToL)
Flight Initiation

As described in section 3.1.2 this is the distance at which a character begins to flee from a perceived threat. For the purposes of this engine it is also the minimum distance that a character will try to maintain between itself and a perceived threat.

Gestation Period

The period from the time of fertilization to birth (Austin, 2004). For example, this is approximately 280 days for humans and 19-21 days for mice.

Reproduction Period

The time range during which the character is capable of reproducing. In humans for example this is approximately from the ages of 14 to 45 or older.

Chance of Reproduction

The percentage chance that a character will produce offspring. For example, a value of fifty expresses that a character will reproduce in half of all sexual interactions.

Chance of Male Offspring

The percentage chance that a character will produce male offspring.

Death After Birth

Whether or not the character dies after birth. (Some organisms die shortly after giving birth such as certain species of spiders and salmon.)

Maximum Age

The maximum age that the character will live until.

Mating Cool-down Period

The period of time that a character must wait after intercourse before they are capable of reproducing again. (This is mostly aimed at male characters who must wait a period of time before producing more sperm.)
Strength

The strength of a character. There is no unit of measurement given to this value as it is meant to be relative across characters. For example, human characters may range in strength from zero to one hundred whereas a dog may range in strength from zero to twenty.

Speed

The amount of spaces which a character is capable of moving in one go. Again there is no unit of measurement given here as it is also relative. A human may be able to move two spaces at a time whereas a turtle may have a speed of one.

Memory

Whether or not the character is capable of remembering encounters and the outcome(s) of those engagements.

Strength/Speed increase/decrease \textit{<per period of time>}

The amount which a character’s strength/speed increases/decreases over a period of time. It is necessary to include this as it can be used to indicate the development of both strength and speed in a character as they mature and to model the decline in strength and speed of a character as they age.

Altruist

Whether or not the character will exhibit altruistic behaviour.

3.3 Communities

There are many definitions for the word community. In biological terms a community is defined as a group of interacting organisms sharing an environment. In ecological terms a community is defined as the assemblage of plants, animals and other living organisms in an area (Rickels and Miller, 1999; Austin, 2004). However, for the purposes of this simulation engine, the term community is used only to describe a collection of characters. A community may contain any number of characters of varying species, though there is a notion of a maximum population number for a community (to prevent it from becoming too large). The reason behind this is to allow for the
creation of complex communities which can in turn allow for the simulation of community interaction through the simulation of character interaction.

### 3.4 Environment

Although character-environment interaction is not being dealt with in this engine, it is still both necessary and appropriate to include some notion of an environment, thus allowing for the introduction of character-environment interaction at a later date.

Having thought about the way in which creatures interact in the real world it is evident that even though some travel great distances, such as migratory birds, most tend to interact with creatures in their local vicinity, especially if they are territorial (Redmond, 2008). Taking this into account, it was decided that the best representation of an environment would be to use a two-dimensional grid where a user can specify the width and height of the environment. Each point on the grid can be occupied by a maximum of one character at a time. It was also decided that the grid would not ‘wrap-around’. That is to say, the distance from the red coloured square to the blue and green squares in figure 3.2 is nine and not one.

![Figure 3.2: 10x10 Environment](image)

There is also an idea of an environment type associated with each environment. While there are a vast amount of environment types in existence, the ones dealt with in this engine are:

- Wetland
The main reason for the inclusion of these types is to allow for the eventual addition of character-environment interaction. As well as this, it is logical that one of the main factors which can affect interaction is the type of environment a character is in. A simple example of this is a frog who is placed in a desert. If the frog is from a wetland then they will be ill-equipped to deal with a desert environment and this will have a direct impact on their performance, and furthermore on their interaction capabilities.

### 3.5 Simulation

A simulation is the name given in this project to the particular simulation which is to be run. A simulation contains one or more communities, an environment and other details. These details include the length of time, in days, that the simulation should run for and the number of loop cycles to represent each day.

Using loop cycles to determine the length of a day allows for further fine grained control over interaction. For example, if twenty four cycles were to represent one day then one cycle would represent one hour. If this were to be scaled up to one thousand, one hundred and forty then one cycle would represent one minute of real time. On each cycle a character can move or interact, thus allowing for greater control over the amount of times characters can interact in a period of time.

### 3.6 Aims

As mentioned previously, in section 3.1.2 under the heading ‘Character Aims’, it is helpful to imagine the aims a character has as a prioritised list.

In this simulation engine a character’s aims have been represented as such a list. Each aim in the list is simply an identifier. The identifier then has an
associated algorithm which will utilise the rules defined for that character in order to attempt to satisfy the aim. A visual example of this is provided in table 3.1.

<table>
<thead>
<tr>
<th>Priority Level</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reproduce</td>
</tr>
<tr>
<td>1</td>
<td>Protect Community</td>
</tr>
<tr>
<td>1</td>
<td>Protect Same Species</td>
</tr>
<tr>
<td>2</td>
<td>Explore Environment</td>
</tr>
</tbody>
</table>

Table 3.1: Visual representation of an aim list

### 3.7 Rules

It is very important to have a rule set for each character as the rules by which a character abides determines how it can interact. Rules can be separated into several categories, depending on what interaction they determine. The categories decided upon for this engine are:

- Attack
- Protect Self
- Protect Others
- Reproduction
- Perceived Threats

The set of rules that can be created should be as diverse as possible. It was decided that the rule sets for a character would take the form of a list of constraints, with the ability to add more constraints to any rule set in the future. These constraints could then be used as a filter, allowing for the use of generic algorithms for interaction. Using these factors, the final high-level design of the rules became:

- A rule for each category previously mentioned
- A list of constraints within each rule
- A list of rules per character
Table 3.2 shows a sample list of constraints within the ‘Reproduction’ and ‘Perceived Threat’ rules for a character.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reproduction</td>
<td>Attributes must be better</td>
</tr>
<tr>
<td>Reproduction</td>
<td>Attributes within $X$ percent</td>
</tr>
<tr>
<td>Reproduction</td>
<td>Age within $Y$ percent</td>
</tr>
<tr>
<td>Perceived Threat</td>
<td>Strength is Greater</td>
</tr>
<tr>
<td>Perceived Threat</td>
<td>Strength is Equal</td>
</tr>
<tr>
<td>Perceived Threat</td>
<td>Has attacked before</td>
</tr>
</tbody>
</table>

Table 3.2: Sample constraint set for ‘Reproduction’ and ‘Perceived Threat’ rules

### 3.8 XML - Representing Components

As the main goal of this simulation engine is modelling character interaction there was no need to worry too much about providing a graphical way of editing the components. Therefore, in combination with the design of the individual components - characters, communities, rules, aims, environments, simulations - XML provides a perfectly reasonable solution.

XML (eXtensible Markup Language) is a markup language which was designed to transport and store data, with a focus on what the data is. It is designed so that tags are user defined and so that it should be self-descriptive when read (w3c, 2011). XML does not actually do anything, it can be thought of as a container for information. The most widely understood API (application programming interface) for manipulating XML is the W3C-approved DOM \(^1\) and there are many libraries across a variety of programming languages which provide parsing and writing of XML files using DOM. Examples include the `xml.dom.minidom` package in Python, `Libxml2` for C and the `xerces.apache.org` package for Java (Minidom; LibXML2; Xerces).

An example of a character defined using XML is shown in figure 3.3 (note

\(^1\)DOM or document object model, is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents.)
that the attributes were generated automatically using pseudo-random values and some attributes have been omitted).

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Character>
  <Name>Human Character</Name>
  <Species>Human</Species>
  <UUID>12345-abcd-6789</UUID>
  <Sex>FEMALE</Sex>
  <Age>35</Age>
  <MaxAge>100</MaxAge>
  <ChangeAttrEveryX>365</ChangeAttrEveryX>
  <DecreaseStrengthAt>223</DecreaseStrengthAt>
  <StrengthPerYearIncrease>0.2</StrengthPerYearIncrease>
  <StrengthPerYearDecrease>0.2</StrengthPerYearDecrease>
  <ReproductionStart>14</ReproductionStart>
  <ReproductionEnd>100</ReproductionEnd>
  <GestationPeriodDays>270</GestationPeriodDays>
  <ChanceRepro>44.7810190456167</ChanceRepro>
  <ChanceMale>85.82883077270532</ChanceMale>
  <Strength>6.11837917851685</Strength>
  <Speed>5</Speed>
  <VisionRange>100</VisionRange>
  <ThreatAvoidance>1</ThreatAvoidance>
  <Altruistic>true</Altruistic>
  <CanRemember>true</CanRemember>
</Character>
```

Figure 3.3: XML representation of a Character

Figure 3.4 shows an XML representation of a character’s rules and aims (again, some rules and aims have been omitted).
It was decided that since characters were being represented using XML and a community contains a number of characters, the best way to represent a community in XML is as a list of file paths to appropriate characters. Figure 3.5 shows an example of a community in XML format.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Community>
  <Name>Community Sample</Name>
  <Description>Community Sample</Description>
  <LastModified>03|04|2011 16:59:48</LastModified>
  <MaxPopulation>10</MaxPopulation>
  <CharacterPath>/some/file/path/character0.char</CharacterPath>
  <CharacterPath>/some/file/path/character1.char</CharacterPath>
  <CharacterPath>/some/file/path/character2.char</CharacterPath>
  <CharacterPath>/some/file/path/character3.char</CharacterPath>
</Community>
```

Figure 3.5: XML representation of a Community

Figure 3.6 shows a sample environment XML file.
Finally, given that a simulation contains a number of communities and an environment, along with other information, it is logical to represent a simulation in XML, in a similar fashion to a community - as a list of included community file paths and an environment file path. Figure 3.7 shows an example of a simulation XML file.

3.9 Displaying Results

Given that there is no Graphical User Interface for this engine it was necessary to provide some other clear means of displaying the outcome of character interaction. It was decided that log files would be the most appropriate mechanism to do so. As XML was decided upon for defining all of the main components in the engine it logically followed that data would be logged to XML files also. This also means that XML log files can be displayed easily in any text editor or browser, and through the use of cascading style sheets
the data can be made clearer and easier to read. Figure 3.8 shows part of an example XML log file.

```xml
<log>
    <record>
        <message>Generating some XML files for use in test Simulation run.</message>
    </record>
    <record>
        <message>Writing Character Human Character 0 to file /home/graysr/CharacterFiles/char0.char</message>
    </record>
</log>
```

Figure 3.8: Snippet of a sample XML log file

### 3.10 Class Architecture

The architecture for the design of this simulation engine is outlined in figure 3.9 (note that ‘attribute*’ and ‘other methods*’ denote attributes and methods which have been purposely omitted).
From this figure it can be seen that each of the individual rules, Reproduction - ProtectSelf, extend the Rule class and that this class contains a list of Constraints. It can also be observed that the Character class has a list of both Rules and Aims, and that the Aim is simply a container that holds an identifier. Furthermore, it can be observed that the XMLWrapper class is used by the Character, Community, Environment and Simulation classes in order to populate the classes from XML files. It can also be seen that a Community object contains a list of Characters and that an Environment simply contains a type along with a width and height. A Simulation contains one Environment and one or more Communities. Finally, it can be
noted that the Engine class contains only one Simulation object and that it has methods to run that simulation, based on the data it contains.

The key points to note about this design is that it is quite modular. The design was purposely built in this way to allow a user to modify any component easily, or to easily add new components.

### 3.11 Engine Algorithm

The main algorithm designed for the simulation engine is shown as a flowchart in figure 3.10.

The basic idea is as follows:

1. Loop through until there are no days left to simulate.
2. Before looping, remove any characters marked for deletion/death.
3. On each loop, run through all of the communities.
4. For each community, run through its characters.
5. For each character, get a list of its aims.
6. Attempt an aim-based movement/interaction based on the current aim.
7. If this fails, move on to the next aim and repeat previous step.
8. If no aim-based movement/interaction is possible, don’t move.
9. Update the character’s age and any other details.
10. If the character has died, either as a result of age or interaction, mark it for deletion/death.
Figure 3.10: Simulation Engine Flowchart
3.12 Movement & Pathing Algorithms

There are two ways in which one can think about a character moving from one point to another. The first is that the character wishes to get to that point and the second is that the character wishes to get closer to that point. Although these sound similar, they are in fact very different. Figure 3.11 shows the differences between the two. In each figure the red square is the current position and the blue square is the destination. If a character wishes to merely get closer to their destination then all full squares under the green arc in figure 3.11a are valid places to stop. However, if the character wishes to get to the destination then the only valid point is the blue square, as shown by the green highlighting in figure 3.11b.

![Figure 3.11: Moving to a point](image)

In order to implement movement to a point and movement towards a point it is necessary to discuss pathing algorithms \(^2\). The reason pathing algorithms are necessary is because one can assume a character will always want to take the shortest path possible to their destination. As well as this, other characters in the environment can be thought of as constraints. It is not possible to move to any occupied point, and a character may not want to move closer to certain other characters. For example, in figure 3.12 the black squares represent occupied points. Here, in order to move from the red square to the blue square, the character must look for a path around these points.

\(^2\)A pathing algorithm is an algorithm which is used to find the shortest route between two points.
The algorithm for movement to a point is as follows:

1. Take the starting point and set it as the current point. Set a step counter to 0.

2. For each of the 8 adjacent squares to the current point, attempt to move to each one in turn, starting with the one closest to the end point. For each point that cannot be moved to, blacklist it.

3. If unable to move to any of the 8 squares then movement was unsuccessful, go to step 6.

4. If you are at the end point, stop.

5. Set the current point as the previous point. Take the point that was moved to and set it as the current point. Increase the step counter by 1. If the step counter is greater than the maximum distance the character can move, go to step 6. Otherwise repeat from step 2.

6. Movement was unsuccessful or the character has moved the maximum amount of steps possible and not reached the end goal. In this case, decrease the step counter by 1 and move the character back to the previous point. Blacklist the point the character was at. Repeat from step 2.

The algorithm for movement towards a point is similar to the algorithm above, except that in this case it does not matter if the character has moved the maximum steps possible and not reached the end goal, as in theory they should still be closer than when they started.
3.13 Loops Versus Threads

Loops were favoured over threads for the design of this simulation engine for several reasons which will be discussed in this section.

The main reason that loops were chosen over threads was due to thread-safeness. One of the biggest problems with parallelism in an application is that one must ensure that the code is thread-safe. If it is not, undesired consequences can occur when several threads make changes to the same set of data, and the result of one thread is changed or overwritten by another.

In the simulation engine, two characters cannot possibly occupy the same point in space (in correspondence to reality). For this reason, if the engine were to be parallelized then each thread would need access to an environment object. If one were to use a local copy of the environment object for each thread then a situation could arise where characters in two threads end up moving to the same point. Then, when the threads finish and the results are put together there would be a need to reorder characters to available locations - an obviously complicated and unwanted process. Another threaded solution would involve having the environment object shared across all threads. If this were the case, then each thread would have to request a lock on the environment when performing reads and writes. This would have the result that threads would have to wait on other threads to finish writing a character’s new position before reading the environment object. This would in effect mimic a series of operations.

Other reasons for favouring loops over threads are that the onus in the simulation engine was the modelling of realistic character interaction and not on performance, and loops are much easier to implement than threads.

Even though the use of loops means that character interaction happens sequentially and not simultaneously this makes very little difference. If movement were to happen simultaneously then in order for character interaction to occur each character would need to process a ‘snapshot’ of their surroundings before moving or interacting. A change in the state of their surroundings while they were processing that snapshot would have no effect on the end result. Similarly, when each character moves sequentially they are using a snapshot of their surroundings in order to determine how they should move or interact. Axelrod and Hamilton (1981) discusses a similar scenario when talking about the implementation of their model for the Iterated Prisoner’s Dilemma. Their model assumes that choices are made simultaneously, how-
ever they state that ‘it would make little difference if there were treated as sequential’.

3.14 Development Model

In this project, the Agile development model was employed.

In this model, an initial working version is designed and implemented as quickly as possible, with very basic functionality. This version may have bugs and may not be 100% operational but it is still useful. After an evaluation of this version the process begins again. The idea being that after each iteration of the process more features are added, producing a new working version. This method minimizes overall risk and allows the project to adapt to any changes very quickly (Shore and Warden, 2007, pp. 6-11).

3.15 Summary

This chapter outlined what was considered in making the application for this project. Initially, there was a discussion of the problem domain which included the governing factors in character interaction. This was followed by a breakdown of all of the components that went into this project and a discussion of how each component was designed and the reason(s) behind that design. There was then an overview of the class architecture, engine and movement algorithms used. This was succeeded by the reason behind favouring loops over threads for this project. Finally the development model employed in this project was discussed.
Chapter 4

Implementation
In this chapter, an overview of the implementation of the simulation engine is given. To begin, the choice of programming language used is discussed. This is followed by a discussion of the development environment. Finally some language specific implementations - logging and XML parsing/writing - are discussed.

4.1 Language Chosen

Java 6 was chosen as the language of choice for this project. There were many reasons for deciding upon Java and these are as follows:

- Java is both platform and architecture independent. This is because Java source code is compiled to byte-code. This byte-code is then interpreted by a JVM (Java Virtual Machine) on the target machine and executed. So long as the target machine has a JVM installed, it should be capable of running a Java application.

- Java is a robust language. It has strong memory allocation and automatic garbage collection mechanisms. It also provides powerful exception handling and type checking.

- Java is an object oriented language. As the class architecture design (discussed in section 3.10) uses both inheritance and encapsulation it was logical to use an object oriented language.

- There are numerous tools available for developing Java applications. Examples include Eclipse, Netbeans and JDeveloper.

- There are a vast number of free, open-source libraries available for an abundance of application requirements.

- Java is a very popular programming language and is widely used from application software to web applications. Many large companies use Java every day for large, complex projects - for example, Murex, IBM and Google.

4.2 Development Environment

4.2.1 Platform

The simulation engine was developed on several machines, each running a different operating system. These were:
• 64-bit PC running Ubuntu Linux 10.04
• 32-bit PC running Windows XP
• 32-bit PC running Linux Mint 7
• 64-bit PC running Debian 5
• 32-bit PC running Debian 4

Most of the development was done on the 64-bit Ubuntu PC and the 32-bit Debian PC. Developing on several machines is helpful in ensuring that code is capable of executing on multiple platforms and architectures.

4.2.2 Eclipse

Eclipse was the IDE (Integrated Development Environment) of choice for this project. Originally developed as an in-house application for IBM, Eclipse has grown to become a widely used and respected IDE (Redmond, 2008).

As Java was the language chosen, any text editor or Java capable IDE could have been used. However, Eclipse was chosen due to its open-source nature, the large number of plug-ins available and the many powerful features it contains.

4.2.3 Backup and Versioning

The backing up of code and documents/files is of utmost importance with any project for several reasons.

• Without back ups, any hard drive failure, theft or damage could result in catastrophic loss and set a project back by days, weeks or even months.

• Regular backing up of work allows for detailed reviews between versions. These are helpful in determining how a project is progressing.

• Backups allow for changes to files to be easily reversed by reverting to a previous version. This is helpful if a change is made in error and then saved.

It was decided that data would be backed up to an external, remote device with backups also kept on the local machine (in case of network issues etc.). The service used for the backing up of data was Dropbox (http://www.
dropbox.com). Dropbox is a web-based file hosting service which utilises cloud computing in order to allow users to store data by file synchronisation. The main reasons that Dropbox were used are:

- Dropbox offers a free service with storage capacity of 2GB.
- Files can be accessed from anywhere so long as there is internet access.
- Dropbox provides a cross-platform application for synchronising and viewing files. This application monitors a specific directory and all of its sub-directories and uploads any new files or changes to files to the cloud. These files remain accessible on the local machine even without internet access.

Subversion was also used in conjunction with Dropbox. Subversion is a software versioning and revision control system. Subversion allows a user to commit changes in files to a database. Any change can then be undone by reverting to a previous version. On the Windows PC Tortoise SVN was used for version control in conjunction with Dropbox. On the Linux machines the command line application SVN was used.

### 4.3 Logging

As discussed in section 3.9 it was decided that results would be logged in XML format. Java provides a very powerful logging manager. The JDK contains the `Java Logging API`. This allows applications to log messages to a central place in order to report errors or provide additional information as to how a program is operating. The package `java.util.logging` provides the logging capabilities via the class `Logger`. Setting up a new logger is as simple as

```java
import java.util.logging.Logger;
private final static Logger LOGGER;
LOGGER = Logger.getLogger(MyClass.class.getName());
```

A logger can also have access to several handlers. A handler recieves the log message and exports it to a certain target, for example to a console or to a file. The way in which a handler operates can be defined through the use of a formatter. `java.util.logging` provides two formatters: `SimpleFormatter` for formatting logs as text and `XMLFormatter` for formatting logs as XML. The `XMLFormatter` provided is not suitable for this project however, as it includes far too much unnecessary information which causes bloating of the log files. To overcome this, a custom xml formatter class, `myXMLFormatter`, was
written for the purposes of this project. This class extends the XMLFormatter class and is used to ensure that only relevant data is logged.

### 4.4 XML Parsing and Writing

The Xerces Java Parser was used in this project for parsing and writing XML files.

In order to parse XML files, the Xerces parser takes in a file and builds an easily traversable tree from the tags in the XML file. The Xerces package provides methods which allow a user to easily search for specific tags within this tree, navigate the tree and extract information.

In order to write XML files, the Xerces package allows a user to easily create a new document and append nodes to this document. Each node can be of several types, such as Element or Text Nodes. This allows a user to create their own node tree and write this to a file.

### 4.5 Summary

In this chapter, an overview of the implementation of the simulation engine was given. To begin, the choice of programming language used was discussed. This was followed by a discussion of the development environment. Finally, there was a discussion on some of the language specific implementations - logging and XML parsing/writing.
Chapter 5
Evaluation
This chapter views the completed project and evaluates it against the aims which were laid out at the beginning of this report. First of all it reviews the project in its entirety and provides a discussion on what elements were completed and which were not during the life-cycle of this project. It then looks at several tests which were performed to judge how well the project was implemented.

## 5.1 The Finished Product

### 5.1.1 Successes

The finished product is indeed a simulation engine capable of modelling character interaction. The engine allows for the definition of characters, communities and environments (through the use of XML files). The engine provides a generic template for character creation, and, through the use of this template, many different types of character can be defined. The results of character interaction are logged using XML files which are easily readable and unambiguous. The time frame for character interaction is user-definable, allowing for interaction to occur in real time or otherwise. Finally, the rules which determine the way in which characters operate are easily defined and edited.

### 5.1.2 Limitations

While reproduction was implemented in this project, it was not implemented as a clear reflection of reproduction in the real world. It is incredibly complex to write reproduction and evolution algorithms at the level of phenotypes, or observable traits. In order to write proper algorithms for reproduction, which in turn affect evolution, it is necessary to break down phenotypes into genotypes. That is to say, reproduction must be done at a genetic level. It was decided that this was outside the scope of this project, as the onus was on character interaction. Therefore, the implementation of realistic interaction algorithms was at the forefront of this project. For this reason, the reproduction algorithms employed are very basic and simply take the average of the parents attributes and apply these to the offspring.

Furthermore, the attack algorithms which were implemented in this engine are also an unclear reflection of real world conflict between animals. While they are relatively indicative of real-world scenarios they can not be considered realistic. This is due to the complexities associated with attack algo-
rithms and the way in which different species deal with conflicts. Maynard Smith and Price (1973) contains an interesting discussion on the topic of animal conflict. They state that ‘conflicts between animals of the same species are usually of “limited war” type’. That is to say that same-species conflicts typically do not result in serious injury. For example, in many species of snakes, the male population wrestle without the use of fangs, and in deer species, the bucks fight without causing injury by locking their antlers. However, there are also many instances in nature where animals attack to kill or seriously injure. This can be seen when two hippopotamuses engage in conflict, as the resulting injuries are often quite severe. The complexities associated with attack are many. How much damage does each character cause? When do characters ‘back off’, become injured or die due to an attack? How much is a character fatigued during an attack? These complexities are noted by Maynard Smith and Price (1973) when they state that in real animals there exist categorical distinctions in attack algorithms across species and also ‘individual differences in the intensity and skill with which each kind of [attack] tactic is employed’. For these reasons, the attack algorithms implemented are only a very simple indication of how real life conflicts work, using a rudimentary points based system to determine when a character is injured, dead or should back off.

5.2 Testing

All tests were performed on a 64-bit Ubuntu Linux 10.04 PC. The technical specifications of this PC are as follows:

- 2.8GHz Intel Core i7 Processor (Quad Core with hyper-threading)
- 6GB (2x2GB + 2x1GB) DDR3 SO DIMM 1333MHz RAM

5.2.1 Interaction

In order to determine whether or not character interaction was successful in the finished simulation engine, several tests were decided upon. These are:

- Can real world data be used?
- Do characters interact and move based on their aims and rules? For example, if a character wishes to mate, will it move towards suitable mates and attempt reproduction.
- Can many characters interact at the same time?
• Can characters of different ‘types’ interact?
• Can characters from several communities successfully interact?

Using Real World Data

Two examples of the effective use of real-world data in the simulation engine are changes in strength over time and gestation periods.

Voorbij and Steenbekkers (2001) performed a study on 750 people in order to determine the changes in strength over time between both males and females. Figure 5.1 shows the results of this study.

![Figure 5.1: Percentage Strength Vs. Age in Humans](image)

In order to show that a similar age vs. strength model could be implemented, several ‘human’ characters were simulated interacting for a period of 100 years, with 365 loop cycles per year (1 cycle = 1 day). The characters were divided into an equal ratio of males and females. The characters were set so that males would reach 100% strength at 25 years and females would reach 65% the strength of males at age 25. Characters were also set so that, by the age of 80, males’ strength would decrease to approximately 70%, and females’ to approximately 40%. The results are shown in figure 5.2 (note
that the data was read from log files and plotted using GNUPlot).

![Percentage Strength Vs. Age in Human Characters](image)

Figure 5.2: Percentage Strength Vs. Age in Human Characters

As can be seen, the results from both are quite similar. It is clear that the simulation engine’s strength vs. age model is a little less fine grained than the actual study, but the results are quite close and can be used without affecting the realism of the engine.

Many species have different gestation periods. The ability to set the gestation period on a per character basis allows a user to use real world figures for individual characters. This is important as an engine where, for example, humans and elephants both had the same gestation period would be almost unusable as it does not reflect real life in any way.

**Aim- and Rule-based Interaction**

To determine whether or not characters moved based on their aims and rule sets, a very simple set of tests were carried out and the output was printed in a very simple text format.
The first test involved two characters ‘a’ and ‘8’ (names determined from a randomly generated UUID), both with similar attributes but of different genders. Each character wanted to mate with a similar character and, as figure 5.3 shows, the characters moved towards each other in an attempt to reproduce.

Another simple test involved two characters in a ‘cat and mouse game’. Figure 5.4 shows the results of this interaction, where character ‘8’ wanted to attack character ‘a’ and ‘a’ attempted to move away in order to avoid injury.
Multiple Character and Community Interaction

A number of tests were carried out to check if multiple characters and communities could interact within the engine. This involved creating characters using pseudo-random data (including species, aims etc.) and populating communities with these random characters. A simulation was then run with ten of these randomly generated communities on a 50x50 environment grid. From careful examination of the log files produced, it was found that characters of different types and from several communities could indeed interact.
5.2.2 Scalability

In order to see whether or not the engine was scalable, several tests were run for varying time periods and with different numbers of characters/communities. The characters were generated using random data and the communities were populated with random characters. The time taken for each of these simulations is listed in table 5.1.

<table>
<thead>
<tr>
<th>No. of Communities</th>
<th>Characters per Community</th>
<th>Cycles per Day</th>
<th>Days to Run</th>
<th>Running Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>12</td>
<td>365</td>
<td>1.227</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>24</td>
<td>365</td>
<td>1.935</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>24</td>
<td>365</td>
<td>16.168</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>48</td>
<td>365</td>
<td>30.346</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>48</td>
<td>3650</td>
<td>315.195</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>24</td>
<td>365</td>
<td>253.045</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>48</td>
<td>3650</td>
<td>5166.787</td>
</tr>
</tbody>
</table>

Table 5.1: Running Times

From these results it can be seen that the slowdown factor is proportional to the number of communities, number of characters, cycles per day and days to run. I.e. if the same simulation is run twice, for 356 days the first time and for 3650 days the second time, it will be approximately ten times slower on the second run. However, it can also be seen that simulating the interaction of 1,000 characters over a period of 3 years, where each day has 48 ‘rounds’ of interaction, takes a mere 5167 seconds, approximately 19,466 times quicker than real time.

5.3 Summary

In this chapter the completed project was evaluated against the aims which were laid out at the beginning. Firstly, it looked at the project in its entirety and what elements were completed successfully and which was not. It then discussed and provided an analysis of several tests which were performed to judge how well the project was implemented.
Chapter 6

Conclusion
This chapter will discuss the conclusions that were reached during and after the development of this simulation engine. First there will be an overview of the project as a whole. This will be followed by a discussion of future work - features which can be added and directions in which the project can go. Finally it will consider everything that was learned throughout the course of the project.

### 6.1 Discussion of the Project

Overall, the simulation engine was successful. The majority of the aims were met, and the core functionality of the simulation engine was developed. Further to this, a great deal was learned throughout the course of this project.

While some aims were not met, due to their unforeseen complexities, they were studied in detail. These complexities formed the basis for discussion and sound conclusion for this project - the main conclusion being that life is not as simple as it may seem. There are hidden complexities involved in every facet of life, and a great deal of variables involved in character interaction. This also gives leeway to an abundance of future paths this project could take in order to further refine the simulation of character interaction based on real world data. However, those aims which were met allowed the project to simulate character interaction based largely on real world data, a task which is extremely challenging and has yet to be implemented in fine grained detail.

### 6.2 Future Work

While the majority of the goals set out at the beginning of the project were met, some ideas were investigated and found to be beyond the scope of the project. This was due to the fact that they were too complex and would have taken too much time to implement. These items could easily be implemented at a later date, given more time. On top of this, there are a number of ways in which the simulation engine could be improved, which are discussed in the following sections.

#### 6.2.1 Environmental Interaction

As mentioned in section 3.2, a notion of an environment was included from the start of the design phase in order to allow for future implementation of environment-character interaction.
A complete implementation of a simulation of environments, along with environmental resources (food, water etc.), would allow the simulation engine to model character interaction and character-environment interaction. This would allow for things such as territoriality, organism distribution across an environment and the modelling of population numbers against resources.

As a notion of an environment was built into the design, this would not be terribly difficult to implement. It should be noted that the only reason it was left out was because the primary goal of the project was to achieve a simulation of character interaction.

6.2.2 Reproduction and Evolution Algorithms

In section 5.1.2, I presented a discussion on the implementation of reproduction and evolution algorithms in this engine.

In order to implement these algorithms to simulate these aspects realistically, it is necessary to take a character’s attributes and convert these into some sort of genetic representation, as it is far too complex to write evolution and reproduction algorithms at as high a level as attributes. One way of doing this would be by representing each of a character’s attributes as a bit string. For example, a strength value of 9 could be represented as ‘01 1001’. This would then allow for genetic algorithms, such as one-point crossover or array crossover (see figure 6.1), to work on a character’s bit string representation of their attributes. It would also allow for the implementation of ‘fitness’ functions - methods which determine the fitness of a gene or individual organism in relation to the rest of the population. (Sywerda, 1989; Rawlins, 1991)
6.2.3 Attack Algorithms

In section 5.1.2, I presented a detailed discussion on the attack algorithms employed in this simulation engine. These algorithms could be improved on and made more realistic by allowing for much more complex rule sets to define attack and the way in which characters engage in conflict.

6.3 Reflection

Prior to starting this project, I had no real understanding of evolutionary processes or the driving forces behind animal interaction, indeed I had no concept of the vast amount of variables associated with such interactions. I also knew very little about simulation engines in general. I am certain that my new found knowledge and understanding of these processes will lend itself to many future ideas and projects. It has already transformed the way I think when I see animals interacting in nature.

Further to expanding my knowledge in the field of natural sciences, I also improved many of my technical abilities and learned how to use new tools. Before beginning this project I had only a very basic working knowledge of
\LaTeX, however, my proficiency in this typesetting language has increased vastly. Moreover, my programming abilities have increased significantly. I have learned many new techniques, some specific to the Java programming language, and others which can be applied to many projects and used across a multitude of languages (such as algorithm design).

This was also the first large solo project which I have worked on. There have been many projects in the past which have been similar in size to this one, however each of this was a group or community project. Implementing a project of this size on my own has given me an invaluable insight into the way in which I work and how I can better my abilities and approach to application design and implementation in future projects. For example, I tend to dive into projects and make unnecessary mistakes. Whilst working on this project this trait resulted in many lost hours where I ran ahead of myself, writing code without properly thinking through the design and project solution, only to have to spend time re-designing and re-implementing. In future, this experience will allow me to increase my productivity and better my solutions.

6.4 Summary

This chapter discussed the conclusions that were reached during and after the development of this simulation engine. First, there was an overview of the project as a whole. This was followed by a discussion of future work, i.e. features which can be added and directions in which the project can lead. To conclude, there was an overview of what was learned as a direct result of this project.
Bibliography


