The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) __________________________
**Summary**

This project aims to simulate certain behaviour patterns of animal species in nature that have inspired engineering algorithms. One species studied is the Cuckoo bird and their breeding behaviour commonly referred to as brood parasitism which has inspired the Cuckoo Search algorithm. The other species studied is Honeybees and their ability to forage for food which has inspired several algorithms that utilise this searching behaviour. The project will be concerned with producing a simulation of these two behaviours and studying different factors that affect what occurs and what choices are made. The report is comprised of an introduction to the problems and history of the area, learning behaviours and a discussion of applications of the original algorithms. Following from this will be the design and implementation of the two simulations concluded with testing and evaluation.
Acknowledgements

I would like to thank my supervisor, Dr Brandon Bennett for his insight and advice throughout the project and his guidance towards the end when I began to struggle.

Thanks are also to be extended to Dr. Marc de Kamps for the motivation he gave me in the progress meeting to be able to tackle the final phase of the project.

And I would like to thank the friends who have stood by me and supported me especially those I have met in the last three years, they have all been the best of me and a lot of them are the greatest people that I have had the good fortune to meet and it is thanks to them that I have endeavoured to get to where I am today.
# Contents

## 1 Introduction

1.1 Aim ..................................................  1  
1.2 Objectives ..........................................  1  
1.3 Minimum Requirements ...............................  2  
1.4 Possible Extensions .................................  2  
1.5 Methodology .........................................  2  
1.6 Schedule .............................................  3  
    1.6.1 Revised Schedule ...............................  4  

## 2 Background

2.1 Overview .............................................  5  
2.2 Cuckoo Search .......................................  6  
2.3 Bees Algorithm .....................................  8  
2.4 Learning Approaches ................................. 11  
    2.4.1 Supervised Learning ......................... 11  
    2.4.2 Unsupervised Learning ....................... 12  
    2.4.3 Distributed Learning ......................... 13  
    2.4.4 Reinforcement Learning ...................... 13  
2.5 Models for Implementation ......................... 14  
    2.5.1 Genetic Algorithms ......................... 14  
    2.5.2 Markov Models ................................. 17  
2.6 Levy Flights ........................................ 17  
2.7 Matthew Trafankowski’s Project ................... 19

## 3 Design

3.1 Overview ............................................. 21  
3.2 Matthew’s Design ................................... 21  
3.3 Methodology Steps .................................. 22  
3.4 Design ............................................... 24  
    3.4.1 GUI ........................................... 24
3.4.2 Welcome Menu ............................................. 24
3.4.3 Cuckoo Search ........................................... 25
  3.4.3.1 Nesting Parameters ................................. 26
  3.4.3.2 Eggs .................................................. 26
  3.4.3.3 Food Sources ....................................... 27
3.4.4 Cuckoo Pseudo Code ................................. 28
3.4.5 Bees Algorithm ......................................... 29
  3.4.5.1 Decision Parameters ............................... 29
  3.4.5.2 Patches and Scouts ................................. 30
3.4.6 Bee Pseudo Code ....................................... 31

4 Implementation .............................................. 33
  4.1 Overview .................................................. 33
    4.1.1 First Phase - Fix existing errors in original product ................. 33
    4.1.2 Second Phase - Building the Interface ............................ 33
    4.1.3 Third Phase - Adding Sprites .................................. 34
    4.1.4 Fourth Phase - Simple Model of Movement ......................... 35
    4.1.5 Fifth Phase - Develop the Genetic Algorithm .................... 37

5 Testing and Results ....................................... 38
  5.1 Overview .................................................. 38
  5.2 Performed Tests .......................................... 38
    5.2.1 Cuckoo Search Testing .................................. 39
      5.2.1.1 Abandon nest ..................................... 39
      5.2.1.2 Destroy egg ....................................... 41
      5.2.1.3 Look after egg .................................... 42
    5.2.2 Bee Foraging Testing .................................... 44
      5.2.2.1 Number of scouts equal to Number of patches ............... 44
      5.2.2.2 Number of scouts less than Number of patches ............. 45
  5.3 Benchmark Tests ......................................... 45
    5.3.1 Rosenbrock ........................................... 45
    5.3.2 Rastrigin ............................................. 46
  5.4 Drawing an Image ......................................... 46

6 Evaluation .................................................. 48
  6.1 Overview .................................................. 48
  6.2 Aim and Requirements Evaluation .......................... 48
  6.3 Evaluation of the product ................................ 49
  6.4 Schedule Evaluation ...................................... 50
Chapter 1

Introduction

1.1 Aim

The aim of this project is to investigate the behaviour of particular animal species that have inspired the development of optimisation algorithms in a bid to improve the formulations of such algorithms or potentially create brand new algorithms. Investigation will be carried out through simulations which will be implemented using a selection of relevant algorithms.

1.2 Objectives

The objectives of the project are to:

- Research the Cuckoo Search algorithm
- Research the Bees Algorithm
- Research learning behaviours to better understand the behaviour of animals in nature
- Research different models and algorithms to be used for implementation
- Produce simulations of the two aforementioned algorithms and incorporate them into the existing interface
- Implement a Genetic Algorithm and observe any benefits produced from the evolutionary process
1.3 Minimum Requirements

The minimum requirements are:

- Extend the existing Shepherding software environment to incorporate the simulations
- Develop the Genetic Algorithms to be used for each simulation
- Implement the Bee Foraging simulation

1.4 Possible Extensions

Possible extensions to this project would include:

- Implement the Cuckoo Search simulation
- Discuss the problem of fish schooling and its benefits for predator avoidance
- Implement and simulate the fish schooling problem

1.5 Methodology

To develop the project with good practice and to a good standard, there are several software development models that can be considered. These models outline different steps to be adopted in the process of developing a software application. The most common models include: the Waterfall model, the Spiral model, RAD (Rapid Application Development) and Software Prototyping.

The Waterfall and Spiral models are the most comprehensive and in depth models, applied in situations where large projects are being developed. This makes the applications of these models inappropriate for this project considering its small scale. Software Prototyping focuses on building several incomplete versions of a software application which can in effect have no resemblance to the finished product. This provides the benefit of feedback to the designer from the users or intended users of the product in the early phases of development. RAD incorporates the prototyping approach as the main concept is to iteratively build software, however a key difference is that one of the main models in prototyping allows for a prototype to be completely thrown out; RAD will just build on a prototype until it is deemed fit for purpose. Prototyping is also better suited for projects where the end product will be required for widespread use by users and would require plenty of interaction between the application and the user(s).

Of the types of methodologies that have been discussed, RAD seems the most appropriate for a project of this size. There is a strong emphasis on iterative design and simultaneous planning and
implementation which are useful elements of this methodology. The iterative design gives rise to
being able to continuously build on the product whilst attempting to remove as many errors as possi-
ble before producing a final product. Simultaneous planning will also allow for extra elements to be
considered during development or to allow us to prepare and produce something completely different
from what was originally planned, providing much more flexibility compared to the other methodolo-
gies.

With RAD, there are 4 main phases to work through in building a software product. These phases
are:

- Requirements Planning
- User Design
- Construction
- Cutover

How the chosen methodology will be adopted in this project will be outlined in the Design and
Implementation chapter.

1.6 Schedule

The planned schedule for the project follows here. The background reading is intended to be com-
pleted by week 5. A presentation has been announced, to be done in front of the KRR research group
and is scheduled to take place during week 3. Preparation for this will be carried out during the same
week.

The mid-project report, will document the work done so far up to the end of week 6 as well as
having the entire first draft of the background literature section. A draft chapter for the final report as
well as the table of contents should be prepared by week 7.

The implementation is scheduled to begin in week 4 and finish in week 10. The implementation
will continue throughout the easter break period. The implementation will also be staggered so that
each stage of developing the product is completed before a new stage begins.

Evaluation will begin in the final stages of developing the product. This is because without starting
it here, there would not be much time left to do it after implementation is completed. There will also
be some investigation into the proposed extension of looking at fish schooling behaviour and possible
implementation, if time permits.
The report writing will be carried out throughout the course of the project, with the delivery scheduled for the final week.

1.6.1 Revised Schedule

This is the revised schedule for the project. Delivery dates were changed for some of the tasks due to rescheduling of events such as the presentation which was due to be delivered in week 4 but was not done until week 7.
Chapter 2

Background

2.1 Overview

There are many phenomena that are observed in nature that can and cannot be explained. The ones that cannot be explained still to this day fascinate those individuals tasked with investigating and observing the phenomenon in question whilst the ones that can be explained, even though they may still fascinate many, have inspired the development of models and algorithms used in many fields, particularly Artificial Intelligence. This inspiration stems from studies of animal behaviour that date back across hundreds of years and while development and progress have occurred, there is still plenty to be explained.

In the field of computer science and Artificial Intelligence, studies in animal behaviour have led to inspiration in the area of optimisation problems. Two particular algorithms that have become widely popular in this area in the last decade are the Cuckoo Search algorithm and the Bees Algorithm.

The former algorithm is inspired by the breeding behaviour of cuckoos in nature whereas the latter focuses on the foraging behaviour of bees. These are two contrasting yet crucial problems for animals in nature as they are necessary for the survival of the species.

While these algorithms have been inspired by behaviour in nature, there has been little research into the grounding upon which the algorithms have been derived. This could be due to scalability in observing the behaviour explicitly such that there may not have been enough resources available to dedicate to such a task. Alternatively the task of deriving the algorithms may not demand a definitive
outline of how the processes in nature actually work. However, what does seem to be apparent is that
through the formulations of these algorithms, assumptions have been made along the way and a lot
of potentially useful information could have been discarded. For instance, in the Cuckoo search, as
outlined in [29] the problem is reduced down to one cuckoo laying just one egg in a random nest but
in nature, some cuckoos have become well versed in imitating other species rather than just picking
random hosts.

It seems there are gaps in these algorithms and how they have been formulated and there is an
argument that these gaps need not be altered, however it does suggest there is room for improvement
and that with investigation, it is possible that better formulations could be produced. Simulating dif-
ferent environments can be a good way to approach this problem. It would be more scalable and can
be better observed than what happens in nature. It is hoped that through developing simulations for
such problems that groundwork can be laid for resolving any discontinuities within the aforemen-
tioned algorithms.

In attempting to resolve these discontinuities and generate the simulations, there will be investi-
gation into different learning methods, to describe why particular actions may or may not occur in
environments and there will exploration into a selection of methods that can potentially adopt in order
to implement such simulations. The investigation into learning behaviours will aid in explaining the
behaviour of some animals in nature (such as how bees forage for food) and will also relate to the
algorithms that will be studied as potential tools for implementation.

2.2 Cuckoo Search

The Cuckoo Search is an optimisation algorithm [29], inspired by a natural phenomenon. The phe-
nomenon in question is the breeding behaviour of cuckoos. More specifically, the breeding of cuckoo
species is very unique and is infamous for displaying obligate brood parasitism [17] whereby a cuckoo
will lay its egg(s) in the nest(s) of other hosts.

The strategy of brood parasitism is in itself a risky task. It is possible that the intruding cuckoo
could be confronted by the host bird who then may fight with the intruder before it can lay its eggs.
While brood parasitism behaviour is not the only method in which cuckoos breed, it is very common
and makes the cuckoo a brood parasite. Brood parasitism does provide advantages in that a cuckoo
can spread its distribution of eggs across several host nests in an effort to reduce predators feeding on
them. Once an alien egg hatches in a host nest, the new cuckoo chick can then feed from the host bird
which will be encouraged by the chick to feed it through rapid calling and leaving its mouth open as
an indicator to the host bird to feed it. Cuckoo chicks may even go so far as to clear the nest of any
other chicks as it continues to grow in an attempt to be the fittest member.

Another advantage that cuckoos have by behaving in this way is that they have become adept in mimicking the pattern of other host bird eggs and they are so good at doing this that in some species in particular, a cuckoo is able to lay an egg that, at least in terms of appearance, will resemble the eggs of the host bird [10]. However, even with this ability in mind, there is always a possibility that a bird will detect and remove any alien eggs from its nest.

Another scenario that may occur from this series of events is that a host bird may detect an alien egg but it may still not remove it from the nest and may choose to build a brand new nest elsewhere. This situation is not uncommon but is more likely in cases when a host engages in conflict with an intruder.

Relating back to its application in engineering optimisation, the goal is for an optimal solution (in the form of one egg) to be obtained. Every egg within the environment represents a potential solution with cuckoo eggs representing new solutions. From this, the aim is to “use the new and potentially better solutions (cuckoos) to replace other solutions that are deemed bad” in comparison to the cuckoo egg [29]. Applications of the cuckoo search algorithm in engineering include the Knapsack problem [29, 28] among others. The aim of the algorithm in such applications is to use the new and potentially better solution in place of not so good solutions in the nests thus the focus of the problem is for the cuckoo species to evolve.

The algorithm itself operates as a metaheuristic. Metaheuristics are in themselves a version of heuristic that searches for an optimal solution from a set of solutions. This behaviour is outlined more explicitly in [29], with the idea that the eggs of a host bird represent a solution and then when a cuckoo egg is laid, it is an improved and thus better solution than those already in the environment.

Metaheuristics are ideal for combinatorial problems and in their original definition, outlined in [8] they were designed to avoid using local optima and searching the entire solution space, further indicating their use in looking for an optimal solution. Although they are designed for finding such optimal solutions, there is no guarantee that it will always find the optimal solution and may still get stuck at a local optimum.

As mentioned previously, the algorithm itself is reduced down and a solution is obtained from only one specific egg, however it may be possible that several eggs can represent a more optimal solution. This possibility does not seem to be considered by Yang in his research so far [29, 28] and as such there is little evidence presented to suggest a method of selection or how one egg is determined to be better than another beyond what is already documented.
There are also aspects of the environment that could have been potentially explored, such as food resources (would there be enough food to support an influx of extra members in the environment?) and the population density would be greatly increased. The emergence of the new individuals in the environment could also greatly affect the survival of other species.

2.3 Bees Algorithm

The Bees Algorithm [7] is another example of an engineering optimisation algorithm that is inspired by behaviour of a particular animal species. In this case, the algorithm is inspired by the foraging behaviour of bees.

Generally, insects show a unique social ability to come together in an effort to achieve tasks such as caring for their young, protecting their hive and searching for food, whilst displaying no obvious signs of communication [23]. Through in depth research into the behaviour of some species it has been found that some insects do have particular methods and abilities in place to be able to communicate with one another.

To look at another foraging model briefly, ant colonies will search for food by sending out a number of ants to find nearby food patches. Ants will traverse great distances to reach food sources and more often than not will not always find the optimal path to a food source (or even an optimal food source). However, over time as ants search more around their neighbourhood, they will begin to narrow the search and eventually find an optimal path to an optimal source. This optimum will be indicated to other ants in the colony by a pheromone which is strengthened as more ants traverse the optimal path.

Bees work in a similar fashion, they too will travel great distances in order to find food patches that may or may not provide good food. There is a more well defined hierarchy within bee colonies of how tasks are distributed. Generally, female bees will work as the “scouts” that will go foraging whilst male bees stay in the hive and work as drones around the hive and look after the young. This is due to the physiology of the species; females are able to collect nectar and pollen whereas the male drone bees are not. Female bees are deemed as workers if they cannot reproduce as reproduction is a task for the queen. Bees also have a unique method of navigating from patches to the colony and back again with two key tools. The first, described in [27] is a ‘solar compass’ which allows a bee to recall where things are located based upon where the sun is. The second, as also outlined by [27] is an ‘internal clock’ that helps a bee determine how far they have flown to reach a patch; this also affords the bee with the ability of keeping track of the sun’s movement which is key for when it returns to the
hive. Once a scout has returned, they will notify onlookers of where to find food based on where the sun is currently rather than where it was when food was found.

There has been plenty of investigation of bee foraging by many individuals and institutions that have attempted to capture, simulate and replicate foraging behaviour. The most recent project in this area, with greatest success, was headed by Peter Bailis and Peter Lifland who worked to produce an extensive simulation environment that considered many more models than will be considered in this project. Their work is documented in [18]. Much like with this project, while they attempted to capture all the parameters that could be important, they could not conceptualise every detail and still found themselves ignoring some aspects of the environment.

A noteworthy aspect of their project is the number of different strategies that were devised in order to perform the simulations and orientate them for specific behaviours rather than just general foraging. All of the strategies are documented in [18] however what is worth highlighting are the selection heuristics for choosing a flower patch. As the paper discusses, when scouts notify their fellow bees in the hive, they do it seemingly at random most of the time [22] and so the heuristics were designed to create some bias to demonstrate the different decisions that bees make. The heuristics that were designed looked at probabilistic choices as well as information of amount of food and its distance - information garnered from the scout performing its waggle dance upon returning to the hive.

Bees will rank how good a patch is based on several particulars including [21]:

- Sugar content
- Amount of food
- Quality

Once bees return to the hive, they will deposit the food that they have collected and will then perform a waggle dance to the members of the hive. This dance informs the hive of particular information regarding [21]:

- Patch Distance
- Patch Direction
- Quality

Once this is done, more bees will be allocated to a patch deemed optimal based on the information provided in the dance. If the patch is optimal more bees will be sent to that patch, however if the patch being advertised in the dance is deemed not good then scout bees will continue to search patches until the optimal is found. As patches are being harvested, bees also have to monitor food levels in that
patch and also evaluate the trade off between the quality of food and the energy required to harvest the patch.

Considering generalised foraging theory, the behaviour of bees when it comes to this task seems to follow the general rules. The main rules are described as assumptions in the literature and it is these assumptions that comprise the general form of foraging models. These assumptions include [6]:

- Decision Assumption
- Currency Assumption
- Constraint Assumption

Each one of these assumptions helps in formulating and designing models for explaining foraging behaviour.

Decision assumptions refer to the type of choices an animal can make or that it is assumed to make rather than a specific choice. Pertaining to models where patch foraging is the focus, the decision is typically represented as a measure of time (i.e. how much time is spent at a patch). An animal will ultimately have to make the decision of leaving a patch to explore other places where it may be able to gather food once it has exhausted the resources available at a particular patch. This choice is strongly influenced by how much energy is acquired from the current patch and as this deteriorates, an animal has to make a choice to search for a source of energy elsewhere before it becomes incapable of doing so.

Currency assumptions are used as a method of comparison between potential values of a decision. This means that a measure exists which determines how much more profitable it is to choose one thing over another. The general concept, outlined in [6], describes the process as supposing a “trait X will exist instead of other traits if X satisfies some existence criterion.” This means that if a particular trait can be observed and it meets what we expect to see, then it will have the most value compared to any other traits. Existence criteria are comprised of two things: a currency and a choice principle. The currency is a reward value whereas the choice principles determine what type of optimisation occurs.

Three types of choice principle exist: maximization, minimisation and stability. Stability occurs when the reward obtained by making a decision depends on decisions of other members in the group. An example of this pertaining to the bees foraging behaviour is when a scout bee returns to a patch it has already foraged, it would have been encouraged to do so if the rest of the hive deem the patch to be good based on the information the scout provides. The reward obtained is that more bees will go to the patch and consequently gather more food for the whole hive.

Constraint assumptions are considered to be limitations of a biological variety. Some limitations lie with the animal themselves and are deemed to be “intrinsic” and some are imposed by the en-
vironment the animals are in, making these limitations, “extrinsic” [6]. Examples of both of these
limitations would be, in the case of extrinsic, as a patch is foraged, the amount of resource at that
patch decreases and an intrinsic limitation occurs from an animal making a conscious decision to not
return to a patch it has already foraged.

Relating back to the application of the algorithm in engineering problems, the algorithm is used
for problems such as: forming manufacturing cells, data clustering and job shop scheduling.

The focus of the problem that will be tackled is based upon bees determining the optimal food
patch(es); in early generations it can be expected that scout bees will not determine the optimal patches
straight away, over generations bees should evolve in a way that allows them to discover these patches
greater.

2.4 Learning Approaches

2.4.1 Supervised Learning

Supervised learning is one of the more notable learning approaches in the field of machine learning.
Derived from the human task, defined in the field of psychology of concept learning [12], supervised
learning is the task of determining something from information that is already provided. This is gen-
erally implemented as some process of classification that will match an input pattern with a specific
output pattern. Another description (in its simplest terms) is provided in [9] that states models in
supervised learning define one set of observations, recognised as inputs and the effect they have on
a different set of observations known as outputs. The information that is provided is known as the
training data and this is typically made up of training examples that will be designed in such a way
that each example is a pair. The “pair” in this sense is the input pattern and its matching output pattern:
an input is going to match to one output.

Supervised learning has more of a focus on the connection between the input and the output which
reduces its applicability to a range of problems as it will be required for us to have these pairs to effi-
ciently utilise this approach to learning. Supervised learning is also not popular for problems that have
a great amount of depth to them [25] meaning the more complicated a problem is, the less efficient
supervised learning is going to be. The applicability of supervised learning is reduced further by the
need for human interaction with the model. A benefit of unsupervised learning is that it can learn from
its own experience and has no human input but supervised learning requires labelling of the examples.

To implement supervised learning effectively for a particular problem, the following steps have to
be followed:
1. Determining the type of training examples: You need to decide what data you want to use as the training set. Directly following from this, the training set can then be gathered. Caution must be taken to ensure such a set represents the problem in the real world, this means inputs and their respective outputs are gathered by some means.

2. The input feature of the function has to be determined. The accuracy of this function will be highly dependent on how the input is represented. The input can be represented through a feature vector which will contain enough information to predict the output correctly. It is important that such a vector is not large and only the bare minimum is used for a reasonably accurate prediction.

3. The structure of the function needs to be determined and an appropriate learning algorithm needs to be applied.

4. The design stage: here the algorithm can be run on the training set defined earlier; some algorithms will require a predefined specification of some kind to determine particular control parameters.

5. Evaluation of the output can finally be performed; evaluation can be done by determining how accurate the function is. After learning, the performance of the function is required to be measured on a separate test set.

2.4.2 Unsupervised Learning

Unsupervised learning is one of the other more common types of learning that have been developed and utilised in machine learning. A much more challenging approach to learning, unsupervised learning is used to deal with cases where there is no labelling to indicate that the inputs will lead to a particular desired output thus removing the necessity of matching inputs to outputs. This sets it apart from the supervised learning approach. This approach is also set apart from reinforcement learning in the sense that there is no desire of obtaining a reward for learning a particular sequence. This approach to learning does provide somewhat of a unique benefit in that it can be applied to many more complex and larger models [25] than approaches such as supervised learning where the focus is on obtaining the expected output from a given input.

This seems to set a different goal for the unsupervised learning approach. Rather than finding the connections between inputs and outputs or obtaining rewards, unsupervised learning approaches aim to build on inputs it receives in an attempt to build models for future tasks such as dealing with new inputs or for decision making purposes.
2.4.3 Distributed Learning

Distributed learning is much more uncommon compared to the other learning behaviours discussed here. Typically, the other learning behaviours can be observed in single agent environments where there is only one individual interacting with the environment it is placed in, however this does not make these learning behaviours exclusive to such environments. Distributed learning on the other hand is exclusive to multi agent environments where there is always more than one agent within the environment which is being observed [26].

The learning in such an environment becomes distributed as all the agents now have to make decisions while making considerations for their counterparts within the environment. Such considerations may include the actions that each agent takes as agents may work differently from each other within the environment. A consideration can also be made for the accumulated knowledge of all the agents within the environment. For example, if the agents need to work together to achieve a goal, then they will need to take advantage of the input that one or many agents may provide to reaching that goal.

A real world example of this would be the problem of ant foraging. As briefly discussed, ants are sent from their colony out to traverse routes to food patches that are within their environment, although this is not a necessary requirement as ants will travel far in an effort to obtain food. Ants will traverse many different routes to reach what they think is an optimal food patch. Optimality in this foraging model is determined more from the distance that ants have to travel to reach food rather than how much is there or how good it is. Each ant is unaware of what every other ant is doing when travelling to a food source and some may end up travelling further away whilst others have found a much closer source.

Once a food source is reached and an ant collects food, it will begin its journey back to the colony whilst leaving a pheromone behind to indicate that food can be found by travelling along that route. This helps other individuals who may not have found a source to move towards discovering one, which helps the collective goal of the hive to obtain food. As more and more ants travel along the route with pheromone placed on it, more and more pheromone will be deposited along that route thereby strengthening the route and increasing the knowledge of each individual, especially those that travelled far from the colony.

2.4.4 Reinforcement Learning

Reinforcement learning is a unique type of learning behaviour. This type of behaviour is set apart from the others previously discussed because of the notion of a reward that is earned through the process of determining actions to be taken by an agent. To build a basic model for this type of behaviour, it would be necessary to have:
1. A set of states describing the environment
2. A set of actions that can be performed at each state
3. Guidelines or some rule set to specify how transitioning between states occurs
4. Guidelines outlining the reward obtained from making a transition
5. Rules describing what the agent observes in the environment

The ultimate goal in a reinforcement learning approach is to learn an optimal policy of actions automatically, where a policy can be seen as defining a sequence of actions to be performed at particular states. As a consequence, a policy must always map the current state to the best possible action according to a predefined fitness measure [13]. This can be a hard task and there are not many techniques that have been developed for this learning approach; one technique that has become widely used is that of Q-learning.

Q-learning works in a way to be expected of the reinforcement learning approach. It takes an action-value function that gives an expected utility of performing an action at a particular state and thereby following a policy representative of this utility afterwards (i.e. if the utility is high, it may be indicative that you are taking the optimal choice of actions). In more specific terms, the utility that is being generated from the action value function is the reward that an agent obtains from performing a particular action at a particular state. The agent is going to have to perform particular actions at particular states in order to obtain higher rewards. The agent will be rewarded at each state based on the action it has taken to get there. Ultimately, an agent is looking to maximise its overall reward from the actions it performs. Before any learning occurs, the utility (or Q value) will be fixed and over time, at each point the state changes, new values will be calculated.

2.5 Models for Implementation

2.5.1 Genetic Algorithms

Genetic algorithms are a class of algorithm developed in a bid to imitate the behaviour of natural evolution. Originally developed by John Holland in the early 1970s, genetic algorithms have now become popular for solving optimisation problems. This is mostly because a lot of behaviour in species follows the mantra of survival of the fittest [13], meaning the best end up surviving whereas the weaker or less preferable members die out. This ideology is perfect for optimisation problems as we are aiming to make the most or best out of a situation. Genetic algorithms are designed as heuristics which
are adaptive (as several natural processes are).

The general form of a genetic algorithm, as defined in [1] is typically as follows:

The Algorithm

- Randomly initialise population
- Determine fitness of population
- Repeat
  - Select parents from population
  - Perform crossover on parents to create a new population
  - Mutate the population
  - Determine the fitness of population

This process is repeated until the best individual is good enough. This is representative of the definition of a genetic algorithm. Generally, we will begin with an unspecified random population which we then determine a fitness from. Then we will select the fittest individuals to create a new generation of individuals from. We then create our new population (process of this described below) and evaluate this new population. The process is repeated, ideally until we reach the optimal solution however if there is some threshold that cannot be invalidated then we may have to settle for sub-optimal solutions.

Genetic algorithms are comprised of 5 key components, outlined in [15]:

- Genetic representation of solutions to the problem
- Method to create an initial population of solutions
- Evaluation function rating solutions in terms of their fitness
- Genetic operators that alter the genetic composition of children during reproduction
- Values for the parameters of genetic algorithms

The process of evolution in genetic algorithms follows three key operations in an attempt to reach the most optimal population. These three operations are:

Selection
At this step, we have already generated a random initial population. At this stage we are evaluating the members of this initial population and determining which of them are fittest to allow us to generate
better populations. The further we evolve our populace, the better it should get, meaning parents that we used perhaps for the first several new populations should be completely phased out by the end of the algorithm unless they were the best candidates to begin with. Fitness of individuals is generally determined by a fitness function.

**Crossover**
The individuals that we have found to be best at the selection phase are then paired off and are designated as the parents of the next new population. This phase is the crossover phase and can be compared to breeding in nature. Crossover is generally done randomly (again to avoid just selecting the optimal solution straight away). Individuals are represented as binary bit strings and their respective values are exchanged on this basis. The offspring created from this process are then added to our next generation. If done correctly, this process of combining and re-combining should and will almost certainly create better individuals for later generations.

![Figure 2.1: A representation of how the crossover operation works as demonstrated in [1]](image)

**Mutation**
At this phase, the new population is modified slightly to maintain some semblance of diversity in the population as well as delaying convergence. This operation is implemented utilising some probability which ideally is kept low so that we can maintain our new generation to some extent without completely altering it altogether. Using binary representation, mutation is generally performed by flipping the bits of some of the new individuals. This can also re-introduce some candidates that could have been lost at the crossover phase [1].

![Figure 2.2: A representation of how the mutation operation works as demonstrated in [1]](image)
2.5.2 Markov Models

Markov models were originally developed towards the end of the 19th century and beginning of the 20th by Andrey Markov. In the beginning, the definition of the term Markov model defined the original idea devised by Markov himself, however this original idea has since spawned not only countless variations of the original idea but also a field all of its own. The original Markov model defined by Markov is now better known as a Markov Chain.

Markov models take on many forms which provide the ability for them to be applied to a variety of problems (some such models will be discussed later) and can be applied in several fields. The area most closely related to Markov models is the field of Probability Theory. The basic model (or rather Chain) is formulated as a consecutive sequence of elements.

Markov models are a form of stochastic modelling and even as the main driving force behind stochastic modelling itself. As described in [9] a stochastic model is a mathematical model or quantitative description of a natural phenomenon. What sets it apart from deterministic models is that the result of the process depends on some measure of likelihood or probability and rather than giving one absolute value, it will give a set of possibilities whereas deterministic methods will only give one output provided it has been given some information to work with.

In a 1st order Markov model, states will only ever depend on things that are learned throughout the model and in a reactive way. This means that the next state within a sequence depends only on the current state regardless of whatever occurred in the past to get to the current state. This changes slightly however when a 2nd order Markov model is used: a new state will depend on the current state and only the immediate state preceding the current one.

Markov models have also become more widely used in the field of Machine Learning which can be expanded further to its relevance in the broader field of learning. The learning approach most closely associated with Markov models is the reinforcement learning approach. As discussed previously, reinforcement learning introduces the idea of obtaining a reward after performing either a particular action or even sequence of actions in order to reach a long term goal. This is where the original Markov model has been expanded to incorporate this reward idea. Two of these extended models are known as: Hidden Markov Models and Markov Decision Process.

2.6 Levy Flights

In the original Cuckoo search implementation by Yang in [29] the algorithm is implemented through the use of a process known as Levy flights. A brief description in the paper outlines an investigation
from a study by [5] that was concerned with the flight path of a particular species of fruit fly. The investigation observed that the general pattern of flight paths were that they were “straight and punctuated by a sudden 90 degree turn”. Although there is not much detail described in [29] there is further discussion pertaining to other occurrences of Levy flights in nature.

Taking a step away from what is outlined in the paper, the original definition of a Levy flight was defined by Benoit Mandelbrot in [14] as one way of defining a particular distribution of step sizes. Levy flights are seen as a form of random walk, however from visual observations, it is easy to see the straight flight paths briefly described in the paper from Yang [29] as well as the sudden turning motions. The distance travelled in a Levy flight is determined by time rather than by actual distance travelled meaning that a very long distance may be traversed before a turn occurs. Examples of this can be seen in the following example graph of a Levy flight.

Levy flights are not uncommon in nature and especially not in the algorithms being investigated. The cuckoo search incorporates Levy flights by simulating a cuckoo flying between nests in an environment. A cuckoo will keep flying around until it has laid its egg(s) and then leaves following a similar flight path.

Levy flights can also be observed in the Bee foraging model. As outlined, bees will scout for good food patches and once one is found they will completely harvest the selected patch and then move onto a new one somewhere else. The flight patterns then follow in a similar style to the Levy flight, however the bees also make extra consideration of where the sun is at the time of finding a patch successfully which may factor into how their flight path looks.

While not a path in the conventional sense, ant clustering also bears some resemblance to levy flights in how they organise themselves within a hive which allows for the application of ant cluster-
ing in methods such as graph partitioning (a discussion of such applications can be found briefly in [16] and more in depth in [4]).

Pertaining to foraging theory, the Levy flight process is not a common model for describing how animals behave when gathering food. While the Levy flight process has provided a good definitive model of movement for the Cuckoo search it is interesting to note that there is not such a model for the bee foraging problem or in fact, for many foraging models. Despite this, in recent years, studies into the development of such models has begun with most progress so far by G.M. Viswanathan, E.P. Raposo and M.G.E. da Luz, with their findings documented in [11] within which they propose the idea that:

“Since Levy flights and walks can optimize search efficiencies, therefore natural selection should have led to adaptations for Levy flight foraging”.

This paper also documents a small selection of example foraging models including honey bee foraging.

2.7 Matthew Trafankowski’s Project

Matthew Trafankowski originally developed a project investigating the problem of shepherding in his final year project [24]. This project considered the collective behaviour of sheep when being herded together by a sheepdog with a particular focus on the flocking behaviour that occurred. Flocking is a phenomenon that occurs in many species in nature with fish and birds being primary examples. In Matthew’s project, he presented extensive research into the problem looking at different formulations for tackling the herding problem with particular reference to the work developed by Reynolds [19] who created the Boids project and provided a good framework for explaining flocking and herding behaviour as well as providing the mechanics for explaining other behaviour types. Matthew also presented a selection of algorithms that dealt with this particular problem.

The key factor common to both Matthew’s project and the one proposed here is the idea of collective behaviour. Within the shepherding problem, collective behaviour emerged from interactions between the sheep within the environment and their desire to move away from the sheepdog. In the Cuckoo Search problem, the collective behaviour that emerges stems from how cuckoos select nests to lay their eggs in and decisions of hosts when an alien egg is discovered. Collective behaviour occurs in the form of co-operation in the sense that cuckoos depend on the hosts to care for their eggs for their species to evolve. The cuckoo chicks also require the host to feed and nurture it until it can leave the nest itself.

A more obvious display of collective behaviour can be observed in the Bees algorithm. Bees have
to go foraging food, gather food (if any can be found) return to the hive and then store it. The bees that have been scouting then have to demonstrate its findings to other bees in the hive and a mutual agreement has to be reached to determine whether other bees will follow.
Chapter 3

Design

3.1 Overview

To be able to simulate the two animal species and their behaviours, a simulation system is necessary to visualise what is happening in each environment for each species. There is already an existing system, built by Matthew Trafankowski [24] for his final year project that investigated the problem of sheep herding. This environment was developed using Java. Considering the environment built by Matthew is orientated specifically towards the sheep herding problem that Matthew was investigating, there will be several parameters and tools within the system that will not be required and some can be ignored altogether. Ideally, the aim is to incorporate both of the new simulations into the system and create new parts of the system that will be specific for the new simulations. There are also some existing bugs in the environment that Matthew built so it will be good practice to ensure that whilst the extensions to the system work, so to must the original system in its entirety.

3.2 Matthew’s Design

From reading Matthew’s project [24] the general design for his project was broken into:

- GUI: This was split into two frames: a parameters frame that functions like a dashboard that can be used for adjusting parameters such as flocking strength, number of sheep, distance, dog speed as well as adding obstacles and additional sheep. The second frame is the area for viewing the simulation itself that allows the user to watch the simulations and how their adjustments to the parameters affect the simulation that is currently running.
• Modules: these were used to separate the different components involved in building the GUI, generating the sprites and animating them to move around in the simulation. Modularisation is good practice for a project of this size, while it is not terribly big it would be inefficient to attempt to build everything for the GUI in one place as well as dealing with the sprites considering there are several different kinds of sprite that need to be used.

• Animator: This is the main class that generates the frame for viewing the simulation. This class does not perform any of the functionality necessary to actually move the sprites but rather focuses on drawing them onto the simulation frame. The functionality for actually getting the sprites moving occurs in the actual sprite classes of each different sprite.

3.3 Methodology Steps

As previously indicated, the RAD approach will be adopted for the development of the software for this project. The steps will be followed in the following:

Analysis and Planning:

In this phase of the project, there is no concern over the implementation itself but rather, focus is on the steps that have to be taken in order to ultimately reach the final product. Additionally, this is where time management begins to become a noticeable issue as not only do the steps in development need planning, but time has to be allocated to each task accordingly to prevent tasks from overrunning, which could lead to rushing towards the end. For this phase, a schedule was drawn up indicating how much time was to be dedicated to each task throughout the course of the project as well as the development of the actual product.

In terms of analysis, some time was spent gaining familiarity with the original product, making practice runs and seeing how changing parameters affected the simulation and how the different parts of the interface fitted together and interacted with each other. This was good to also identify areas of the existing product that needed improving and to establish how the product being produced would be linked in with the existing one. The main steps to be considered at this phase are:

• Review the existing product: what it can already do; what needs to be improved
• Schedule of work for development of product
• Design of interface for Bees Algorithm
• Design of interface for Cuckoo Search
• Design a simple simulation as a prototype
• Development of the Genetic Algorithm

Whilst being a section of its own, it is also good to plan how to test your product so that you know exactly how the product will be evaluated.

Design:

Considering there is no intention of making the system widely accessible to be used by many users, there will not be any necessity in orientating the software for end users. The software still needs to be presentable and fitting for its purpose. Ideally, the environment needs to be relatively simple and still have the right amount of functionality necessary to allow a user to interact with the system and view the simulation. Another consideration to be made is that because the existing system is to be extended, the changes that are made should be done so while ensuring that overall, the system remains the same and that the extensions fit with the existing product (i.e. it would be bad practice to design a completely different user interface and then combine it with what is already in place). This stage begins laying the ground work for the key parts of the development, while nothing is properly built or implemented it is useful to draw up rough ideas for how things will take shape. As can be seen later in this chapter, there are several images highlighting the early designs of the system to be implemented. Also there is pseudo code provided for the Genetic Algorithm in each problem to provide an understanding of how the simulations will run.

Construction:

This phase is concerned with the development of the system itself. The first main step to this will be to build the parts of the GUI that will be necessary for the simulations. Development of the GUI can be handled using an environment that can be used to build an interface easily (such as Netbeans). This can allow for more focus on the development of the simulations themselves. Once the GUI is developed, the next phase will be to implement a simple simulation within the environment that is built along with the GUI for the control parameters. There will not be any functionality behind the control parameters at this stage as the intention here is to just build a simple simulation and observe it running as a standalone program without anything being changed.

This is to ensure that the program can handle displaying and move one or two sprites to begin with before being flooded with many of them. Once this is done, a more improved model will be developed to capture and better represent the actual behaviour that should be occurring. Beyond this, the final part of construction will be to implement the Genetic Algorithm for each simulation. While the core performance of the Genetic Algorithm is going to be the same for both simulations, each one will be designed with their respective purposes in mind: foraging behaviour of bees and breeding behaviour of cuckoos.
Testing:

This is the final phase of the development, moving away from design slightly, this step focuses on actual implementation of the product. Implementation will only occur once a product is produced that either fits the exact requirements set out at the outset of the project or at least satisfies some level of completion. What should be obtained from these simulations are parameters that can be incorporated into a separate program that runs a GA to draw an image using triangles. These parameters should indicate the population with the best fitness and the size of the optimal population. There are also benchmark tests used for Engineering Optimisation that can also be considered to test against.

3.4 Design

3.4.1 GUI

The GUI will be designed with the existing system in mind so that there are no stark differences between the appearance of what is already there and what will be added.

To create some separation between the different simulations and produce more of a generic simulation environment, a welcome screen will be developed welcoming a user and inviting them to select which simulation they want to run rather than having to see everything at once they can then just view what they want each time.

The GUI itself will be developed using the Netbeans IDE. The reasoning for selecting this is that it provides a storyboard like design environment that allows a user creating a GUI to plan how they want it to look and observe the product as they are building rather than just using code and hoping for a good output once it compiles.

At this design phase it should be noted that the proposed designs were altered slightly throughout development but for the most part, retained the appearance that has been outlined here.

3.4.2 Welcome Menu

This screen will tell the user what they are using and allows them to pick the simulation they want to run through selecting a checkbox. Upon selecting a checkbox, the simulation could then automatically be launched or alternatively, it may be convenient to add a button to this screen that the user has to click that confirms their selection and notifies them that they will now be taken to the simulation they have selected. Additionally only one simulation should be allowed to run at a time, thus when a simulation is selected, until the user deselects that choice they should be unable to pick any others.
3.4.3 Cuckoo Search

This section discusses the development of the GUI that will be necessary for the running of the Cuckoo Search simulation. The main parameters to be considered include:

- How likely a cuckoo is to pick a nest
- Food available to members of the environment
- How big the population can be before it can no longer be sustained by the environment

The animator for this simulation will display an environment that will be filled with an initial population of nests. As the population grows and cuckoos evolve, some nests will be abandoned and should disappear from the environment and new nests will be built. The animator will also display food sources in the environment as the population grows; these food sources should dwindle and whilst other food sources may become available later on, these depleted sources should not replenish at any time.
3.4.3.1 Nesting Parameters

![Diagram of Nesting Parameters Panel]

Figure 3.2: Initial design for Nesting

This panel is the main parameters panel for the cuckoo search simulation. This panel will allow the user to change the rate of a cuckoo bird picking a particular nest belonging to one species over another. This allows the user to simulate the behaviour of brood parasitism that cuckoos adopt in nature. As mentioned previously this can help lend the simulation to the problem of conservation as well as leading to development of a better formulation of the existing cuckoo search algorithm.

3.4.3.2 Eggs

This panel will not feature much more than checkboxes indicating how many eggs there are in a nest and how many eggs a cuckoo lays at a time in a nest. This adds further scope to the brood parasitism behaviour of the cuckoo species. Generally, cuckoos only lay one egg in a nest at a time as they are spreading eggs around and so by laying one egg in a nest with many eggs already they are reducing the likelihood of their egg being detected by a host bird, however if a cuckoo lays one or more eggs in a nest with few or even no eggs inside then the chances of the egg being detected become a lot higher and this results in the evolution of the species being adversely and dramatically affected.
3.4.3.3 Food Sources

This panel, at this stage of development, will not allow for additional food sources to be generated or for existing ones to necessarily be replenished but will be used purely for information purposes. The list will tell the user how many food sources there are and allow the user to click on each one. When a user clicks on a food source in the list, information regarding that food source will then be displayed in the text field telling the user how much food is left, how many nests are within a given range and should indicate when a food source has been completely exhausted.
3.4.4 Cuckoo Pseudo Code

Below is the general structure of how the code for the Cuckoo search simulation will be formed. The phases that will tie in with the GA will be: GENERATION, SELECTION and EVOLUTION. Other parts of the structure are still necessary for capturing the full extent of what happens when cuckoos actually breed in nature.

{Cuckoo Search}

INITIAL ENVIRONMENT:
1: Create the initial population of nests in the environment
2: Populate a selection of these nests with eggs
3: Create the food sites in the environment
4: Allocate resource pool for building new nests

GENERATION PHASE:
1: Generate the Cuckoo bird to fly into the environment
2: Cuckoo performs Levy flights procedure before laying an egg

LAYING AN EGG:
1: Cuckoo lays an egg in a nest and leaves the environment
   If nest is unoccupied by host
      Lay egg
   Else
      Pick another nest

DETECTION PHASE:
1: Host bird returns to nest after brood parasitism has occurred
2: A predefined probability of finding an alien egg
   If an alien egg is detected
      Throw egg away
   Else
      Leave nest
   Else if egg not detected
      Remain at current location

SELECTION PHASE
1: Find nest with best fitness - //nests with cuckoo eggs have higher fitness
2: Rank remaining nests - //ranked according to ones still occupied by host
3: Remove the worst ranked nests below a defined threshold:
   If egg thrown away
Nest ranking is reduced
Else if nest abandoned
Nest dies out

EVOLUTION PHASE
1: Surviving cuckoos hatch
2: New nests added to environment if nests were abandoned in previous generation and
algorithm repeats from GENERATION PHASE

3.4.5 Bees Algorithm

This section outlines the development of the GUI required for the running of the Bees problem. The
focus of this is to observe how the decision making process for selecting a patch works and if it mat-
ters to a bee that it ranks a patch based on all three decisions, outlined in Section 2.3 or whether it can
do quite well with one or two.

3.4.5.1 Decision Parameters

The decisions panel will be the main panel for the Bees simulation that will greatly affect the running
of the simulation. As outlined, Bees depend upon three decisions in order to pick an optimal food
patch. As the population evolves in the simulation, ideally bees should use a scheme where they use
all three decisions, however in the beginning it is possible that they only use one and so over the
course of evolution of the colony, it is hoped that they will reach a scheme where they successfully
use all three decisions when it comes to picking a patch. Another approach to evolving the population
will be to have the user dynamically control the dependency on a particular decision using the number
sliders. For example, it may be possible that bees will rank patches as a better option based on the
food quality regardless of the patch being hundreds of metres away from the colony or they may prefer
a patch within a few yards as opposed to a patch that has brilliant food and plenty of it but maybe a
kilometre away. The lists on this panel will break down the main list of patches into separate ones
based on their rankings in a given decision range (i.e. a patch that is in close proximity to the colony
will appear high on the list of patches in direction but may have a terrible quality so will appear quite
low on the quality list).
3.4.5.2 Patches and Scouts

This panel will not involve as much user interaction as the previous one. The lists here will work in a similar fashion to the list structure in the cuckoo search ‘Food’ panel and the lists in the ‘Decisions’ panel. They will serve as a way of informing the user about information regarding the patches and the scouts. Some kind of indication will have to be implemented for the list of scouts to determine when a scout is travelling to a patch and travelling back from a patch.

The reason to have the list of patches is to allow the user to monitor what patches are being favoured by the bees, it should also work to indicate when a patch is completely foraged by which point the bees should have left that patch altogether and should be foraging elsewhere, if some bees are found to still be investigating this patch then this can be useful to detect said bee and help in trying to get the bee to move away from the patch.
3.4.6 Bee Pseudo Code

The code outlining the intended structure for the Bee foraging simulation follows. Like with the Cuckoo search simulation, some phases do not directly tie in with the genetic algorithm. The phases in question are the SCOUTING and DANCE phases. These do however directly effect the fitness ranking of the individuals based on their performance.

(Bees Algorithm)

GENERATION PHASE
1: Place hive at random in the environment
2: Create and place the initial patches in the environment
3: Generate the initial scouts

SCOUTING PHASE
1: Scouts fly away from the hive to locate patches following the Levy flight model
2: Once scouts find a patch, they return to the hive
   If no patch is found
   Bee continues to scout
   If Bee runs out of energy
   Bee dies

DANCE PHASE
1: Successful scouts return to hive and deposit a sample of food
2: Scouts then perform ‘waggle dance’ to inform onlooker bees of findings
SELECTION PHASE
1: Rank fitness of bees based on patch that is found
   If patch is deemed to be good
   Ranking improves
   If patch is not good
   Reduce ranking
2: Remove bees with a fitness ranking below a specified threshold

EVOLUTION PHASE
1: Crossover the chromosomes of the fittest bees
2: Mutate the new chromosomes for a brand new population
3: Repeat from SCOUTING PHASE
Chapter 4

Implementation

4.1 Overview

The nature of RAD, as already discussed is to go through several iterations of development to build up the final product, progressively building on what was built at the previous phase. The parts of the implementation completed at this point have been completed in line with the chosen methodology and are documented, phase by phase in this chapter.

4.1.1 First Phase - Fix existing errors in original product

Before any development could be carried out, revisions had to be made to the existing product because of errors that occurred when the interface was being used. The main issue was that when altering the number sliders to change some of the parameters, the text describing them would disappear. Before development of the new interface began, this was a good first step into also understanding the layout managers provided by Java to get a clearer idea of how the interface would appear. While some attempt was made to fix this, more focus was given to the development of the new interface which, even though this step had been taken beforehand, did not resolve all the issues that occurred. This could be observed as more of a fault on the part of Java. Suggestions for working around this issue are suggested in the following section 4.1.2.

4.1.2 Second Phase - Building the Interface

The first phase of the development was to build the interface that would go alongside the simulation environment itself. One method to work around this could be the use of threads for drawing the inter-
face to be able to repaint the information when it disappears.

The designs outlined in the previous section were adhered to and once several panels were developed, building others was not too difficult a task. The problem in the beginning of development was getting used to the different layouts available in Java. The interface also had to resemble the existing product to avoid difficulties of having to get used to a different interface and retain familiarity when using the different simulations.

One additional part that was added to the interface was a ‘Sprites’ panel that was incorporated in the original product. This was added after the original ‘ViewSprites’ class was modified and implemented to create the Sprites and get them appearing in the simulator window. The Sprites panel, as it is, allows for a user to add and delete sprites from the simulation as it is running. This is not a feature that is intended to be part of the newly developed simulations because they run using a genetic algorithm, the generation of new sprites and removal of old ones has to be automated in the simulation and is the one aspect of the simulations that a user should have no control over at all.

The final appearance of how the interface looks can be seen in Appendix D - GUI Screens

4.1.3 Third Phase - Adding Sprites

Once the interface was developed and the simulation window was in place, the next step was to actually get sprites to appear within the simulation environment. Like with the interface, there was an extensive effort made to ensure that the process of creating and adding sprites to the environment was close to the original implementation. This was difficult as it meant there were limitations on how to tackle the implementation. On the first attempt, the implementation was done following this structure, however there was very little documentation provided with the original code so gathering an understanding of how the structure worked posed a challenge. Once this was achieved a basic structure based off the original code was built and implemented. This first attempt ultimately failed to produce a result, while most of the original structure had been retained, the sprites failed to appear.

The second attempt followed an approach from a tutorial that created a simple animation [3]. This was difficult to adapt to fit the structure that was already in place as it would have meant the original structure would have to be completely scrapped in favour of utilising parts of this animation. The simulation environment that was built would have also needed to be scrapped as this animation was set up to use an entire screen rather than being used as part of an interface. While sprites did appear, because of the difference in the structure of this animation development of this attempt was not extended much further.

The third attempt which ultimately proved the most fruitful was to hard code the individual sprites
into the ‘BeeColony’ class, within a BeeColony function and then from here, the BeeColony class called this function. This was done as a quick fix to resolve the issue, however it has been the best method and with the time left available, is the only one that can be developed more. This also means that mapping the genetic algorithm to fit the simulation would be much harder as the process of generating and deleting sprites should be a dynamic process, changing over time yet the sprites are fixed thus negating this capability.

This method was extended to be applied to the Cuckoo search simulation also.

4.1.4 Fourth Phase - Simple Model of Movement

Extending from the implementation that involved hard coding the sprites into the environment, a simple model of movement was then developed. Ultimately, the simulations have to also encapsulate the Levy flight process, discussed in section 2.6 particularly the Cuckoo search simulation. The process is also applicable to the Bee foraging simulation. The model implemented works by moving the sprite randomly and rapidly in one of four directions. The distance travelled in any one direction is not typically far but extensions can be made to this implementation to be able to incorporate the Levy flight process.

The bee simulation is much closer to the behaviour that is expected to be observed as there is a noticeable hovering motion which is moving rapidly, over short distances in several directions. The cuckoo simulation is not currently representative of how it should work but has captured the simple concept required to build from.

To extend this model to work in fashion similar to the Levy flight process, the maximum distance of travel needs to be increased. This also needs to be randomised because a Levy flight operates like a random walk so the distance travelled before changing direction is undetermined before the simulation begins. The angle of changing direction is typically always 90 degrees and these are certainly the most apparent changes in direction with there being no change in direction greater than this.
Figure 4.1: The environment of the bee foraging simulation
4.1.5 Fifth Phase - Develop the Genetic Algorithm

The next phase of development focuses on the creation of the genetic algorithm. The approach taken to this was rather than building the full genetic algorithm that would be applied as part of the simulation, it was better to develop a Genetic Algorithm that performed a simple task to understand the theory behind developing one in Java. From here, the aim was then to take this simple genetic algorithm and map it to fit in line with the simulations.

The genetic algorithm that has been developed operates by generating a target binary string from a randomly generated string by performing the operations of the GA.
Chapter 5

Testing and Results

5.1 Overview

This chapter documents the testing that will be performed on the product that has been developed and the results obtained from this testing. Additionally, there will also be a discussion of tests that were planned to be carried out once the entire product was complete. The tests are split into two main sections. The first section will document the tests that have been completed and the results of these tests as well as some small discussion of what is concluded from the results. The second section will document the tests that were planned to be performed once the Genetic Algorithm was developed and incorporated into the simulations themselves.

This second section is also designed to relate back to the aim of this project which was to attempt to show that the original formulations of the algorithms could be improved if a Genetic Algorithm was applied. The difficulty of this though is that there were many more unforeseen parameters that had to be considered for the algorithm to work, thus conceptualizing these algorithms in such a manner has been troublesome.

5.2 Performed Tests

This first section of tests, tested the two simulations with the capabilities that they have at this point. For the cuckoo simulation, there were three different behaviours that comprised the entire environment and were tested separately. These were:
• Abandoning the Nest: Host discovers the cuckoo egg and leaves to build a new nest somewhere in the environment

• Destroying the Egg: Host discovers the cuckoo egg and destroys it

• Keep the Egg: Host discovers the egg and decides to look after it, believing it to be one of its own

The behaviour that occurs depends on the cuckoo actually laying an egg and the host birds returning to the nest and actually discovering the egg. For the purpose of the simple simulation, the nests only ever contain the one cuckoo egg that is laid there, if the cuckoo is successful. If the host birds return to their nests before the cuckoo lays an egg, then the cuckoo is prevented from laying any eggs whatsoever.

The movement of the members of the environment was the key determining factor to success or failure for each member and as such, the tests are performed by altering the distance measure. The model of movement is an attempt to model the Levy flight process and the distance measure is used to indicate the minimum distance the bird or cuckoo should travel before making a turn in a different direction.

The Levy flight process was also used in the simulation of the Bees. This was a much simpler approach then trying to model how bees actually forage which is based on the movement of the sun.

All of the tests were also timed to last no longer than ten minutes. If a cuckoo failed to lay an egg or bird failed to forage for food successfully in this time, then both were deemed to have failed. This time limit was also applied to the bee foraging.

Due to the tests being set to run for ten minutes each, the results that were obtained and documented here were gathered after just one test run for each case.

5.2.1 Cuckoo Search Testing

5.2.1.1 Abandon nest

These are the results of testing the NewNest function. The cuckoos had five different distance measures whilst the hosts had three: 20, 50 and 100. This was done to cause some variation between the two species and have them behaving differently rather than moving in formation. All of these distance measures are scaled for the purposes of the simulation and as such, the ratio of distance is one pixel in the simulation is equivalent to one metre. The tests are organised in ascending order of the distance to travel for the host birds.
**Host distance of travel set at 20**

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>2 minutes, 13 seconds</td>
</tr>
<tr>
<td>40</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>2 minutes, 44 seconds</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>6 seconds</td>
<td>Yes</td>
<td>2 minutes, 40 seconds</td>
</tr>
<tr>
<td>80</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>3 minutes, 55 seconds</td>
</tr>
<tr>
<td>100</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>8 minutes, 59 seconds</td>
</tr>
</tbody>
</table>

These first results show that the cuckoo has very little success in being able to lay an egg when the hosts do not have far to travel. The one successful occurrence of the cuckoo laying an egg can be attributed to the random point that the cuckoo entered the environment and the close proximity of a nest.

**Host distance of travel set at 50**

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Yes</td>
<td>39 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>40</td>
<td>Yes</td>
<td>57 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>4 minutes, 41 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>80</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>4 minutes, 39 seconds</td>
</tr>
<tr>
<td>100</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>3 minutes, 50 seconds</td>
</tr>
</tbody>
</table>

These results showed better performance for the cuckoo when the host birds had to travel further. In the first three cases, two of which the cuckoo travelled a shorter distance than the hosts, and the third, the cuckoo was only travelling a slight distance more. The hosts also seemed to be less successful at gathering food in the cases where the cuckoo laid an egg providing the cuckoo with the benefit of having more time to find a nest and lay an egg.

**Host distance of travel set at 100**
<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>3 minutes, 45 seconds</td>
</tr>
<tr>
<td>40</td>
<td>Yes</td>
<td>3 seconds</td>
<td>Yes</td>
<td>1 minute, 7 seconds</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>3 minutes, 30 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>25 seconds</td>
<td>Yes</td>
<td>55 seconds</td>
</tr>
<tr>
<td>100</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>

In this last set of results, the cuckoo fails in the first case to reach a nest, perhaps to its poor placement when it arrived in the environment and the hosts were generally successful in gathering food except in cases 3 and 5. In case 5, when both the hosts and the cuckoo have to travel at least 100, neither species has much success.

5.2.1.2 Destroy egg

These are the results of testing the OldNest function. This function demonstrates the behaviour that occurs when a host bird discovers a cuckoo egg and decides to destroy it.

**Host distance of travel set at 20**

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>2 minutes, 23 seconds</td>
</tr>
<tr>
<td>40</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>2 minutes, 14 seconds</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>1 minute</td>
<td>Yes</td>
<td>2 minutes, 28 seconds</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>20 seconds</td>
<td>Yes</td>
<td>54 seconds</td>
</tr>
<tr>
<td>100</td>
<td>Yes</td>
<td>13 seconds</td>
<td>Yes</td>
<td>2 minutes, 30 seconds</td>
</tr>
</tbody>
</table>

In this first set of results, the cuckoo has actually garnered more success when travelling at greater distances compared to the previous function. The host birds also seem to average around 2 to 3 minutes when travelling at this short distance.
Host distance of travel set at 50

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Yes</td>
<td>3 seconds</td>
<td>Yes</td>
<td>2 minutes, 12 seconds</td>
</tr>
<tr>
<td>40</td>
<td>Yes</td>
<td>1 minute, 29 seconds</td>
<td>Yes</td>
<td>5 minutes, 30 seconds</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>51 seconds</td>
<td>Yes</td>
<td>2 minutes, 6 seconds</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>17 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>100</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>3 minutes, 15 seconds</td>
</tr>
</tbody>
</table>

In this set of results, the cuckoo is successful every time, except for the final case. The host birds are successful at gathering food except for the fourth case.

Host distance of travel set at 100

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>26 seconds</td>
</tr>
<tr>
<td>40</td>
<td>Yes</td>
<td>1 minute</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>60</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>25 seconds</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>2 minutes, 33 seconds</td>
<td>Yes</td>
<td>5 minutes, 34 seconds</td>
</tr>
<tr>
<td>100</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>

In this last set of results, there is much more of a variety to the success and failure of each species in the environment. I cannot definitively describe why this could be but my hypothesis is that due to the turning in each case and the placement of food sites and nests, both the cuckoo and the host birds at times found themselves circling small regions with none of these sites nearby.

5.2.1.3 Look after egg

These tests looked at the LookAfterNest function which demonstrated the behaviour that occurs when a host bird finds a cuckoo egg and decides to look after it, believing that it is in fact, one of its own eggs.
<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>40</td>
<td>Yes</td>
<td>4 minutes, 17 seconds</td>
<td>Yes</td>
<td>5 minutes, 43 seconds</td>
</tr>
<tr>
<td>60</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>7 minutes, 55 seconds</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>2 minutes, 2 seconds</td>
<td>Yes</td>
<td>7 minutes, 41 seconds</td>
</tr>
<tr>
<td>100</td>
<td>Yes</td>
<td>31 seconds</td>
<td>Yes</td>
<td>2 minutes, 3 seconds</td>
</tr>
</tbody>
</table>

This set of results have been the most intriguing to obtain as there is a mixture of success and failure making it hard to judge whether the shorter distance of travel for the hosts is a benefit to the cuckoo. There are more cases where the cuckoo is successful but when it has to travel at the same distance and three times the distance of the hosts, it does not do so well. In the first case, the hosts were also unsuccessful at gathering food within the specified time limit, which lends itself to the indication that the placement of nests and food sources may have been a big factor for this series of tests.

**Host distance of travel set at 50**

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Did Cuckoo lay an egg?</th>
<th>Time to lay egg</th>
<th>Did all birds collect food</th>
<th>Time taken to collect food</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Yes</td>
<td>18 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>40</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>30 seconds</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>1 minute, 53 seconds</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>80</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>1 minute, 35 seconds</td>
</tr>
<tr>
<td>100</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>

In this set of tests, at least one of the two species fails in every case. The hosts were not successful in case 1 and case 3 while the cuckoos did not find a nest quickly enough in cases 2 and 4. In case 5, neither were successful.

**Host distance of travel set at 100**
This final set of results shows yet another mixture of success and failure. This time, whenever the cuckoo has failed in laying an egg, the hosts have also failed to gather food within the time limit. Also, after the second case, the host fails in all other cases following from this, indicating that in this series, the longer distance of travel was a hindrance for the hosts. The cuckoo also fails when travelling at longer distances also with the exception of case 4, however, even in this case the cuckoo has still taken a considerable amount of time to lay an egg.

### 5.2.2 Bee Foraging Testing

#### 5.2.2.1 Number of scouts equal to Number of patches

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Were all bees successful?</th>
<th>How many were successful?</th>
<th>Time taken for all successful bees</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>No</td>
<td>One</td>
<td>30 seconds</td>
</tr>
<tr>
<td>40</td>
<td>No</td>
<td>One</td>
<td>2 minutes, 1 second</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>All bees</td>
<td>3 minutes, 57 seconds</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>All bees</td>
<td>2 minutes, 53 seconds</td>
</tr>
<tr>
<td>100</td>
<td>Yes</td>
<td>All bees</td>
<td>7 minutes, 26 seconds</td>
</tr>
</tbody>
</table>

This first set of results shows that the bees did have some difficulty in finding a patch within the given time limit although one bee was successful, perhaps due to the initial direction of travel it embarked on when leaving the hive and the close proximity of a patch in this chosen direction. From case 3 onwards, however all the bees proved to be successful.
5.2.2.2 Number of scouts less than Number of patches

<table>
<thead>
<tr>
<th>Distance to travel</th>
<th>Were all bees successful?</th>
<th>How many were successful?</th>
<th>Time taken for all successful bees</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Yes</td>
<td>All bees</td>
<td>9 minutes, 10 seconds</td>
</tr>
<tr>
<td>40</td>
<td>No</td>
<td>Three</td>
<td>3 minutes, 25 seconds</td>
</tr>
<tr>
<td>60</td>
<td>Yes</td>
<td>All bees</td>
<td>6 minutes, 16 seconds</td>
</tr>
<tr>
<td>80</td>
<td>Yes</td>
<td>All bees</td>
<td>1 minute, 44 seconds</td>
</tr>
<tr>
<td>100</td>
<td>Yes</td>
<td>All bees</td>
<td>4 minutes, 56 seconds</td>
</tr>
</tbody>
</table>

Aside from the case where one bee failed, the other tests showed that the bees were highly successful when the environment was inundated with patches that they could forage from.

In closing, these initial results have shown some intriguing aspects of the Levy flight process, especially for the Bee foraging model in that, the Levy flights are not utilised by bees when they forage for food but yet the application of a simple formulation of the process here has shown that it is very beneficial for the bees.

With more time for investigation, more tests could have been run for each case and thus would have allowed for an average of all tests rather than being dependant on just one result.

5.3 Benchmark Tests

This section discusses the proposed test functions that would have been used had the application of the Genetic Algorithm been successful.

5.3.1 Rosenbrock

This function has a non-convex design and has been developed as a standard performance test for optimisation algorithms, developed by Howard H. Rosenbrock [20]. The global minimum of the function is found within a parabolic shaped flat valley which is not problematic to locate. To reach the minimum itself however does cause problems and the function itself is defined by:
In the paper [28] Yang defined a stochastic extension, designed as a heuristic and is designed for testing the original Cuckoo Search algorithm as defined in the same paper.

5.3.2 Rastrigin

Like the Rosenbrock function, the Rastrigin function is non-convex and developed for the purpose of benchmarking optimisation algorithms. It was designed by Rastrigin as a 2-dimensional function. The function is deemed to be quite complex due to the large search space and the number of local minima means that finding the optimal solution is difficult as the function is much more likely to stick at one such minima without considering others. The function is defined by:

\[
    f(x) = An + \sum_{i=1}^{n} [x_i^2 - A \cos(2\pi x_i)]
\]

5.4 Drawing an Image

This test is set apart from the others that have been previously described. The idea here is to take the parameters from the simulations that led to an optimal solution. These parameters would then have been plugged into a program created in Python, by Derek Magee, that would have attempted to re-create an image by drawing triangles. The two parameters that would have values plugged into them would be: the population size and the number of generations. The population size would correspond with the number of cuckoos that successfully breed in the cuckoo search algorithm and the number of bees in the colony in the bee foraging problem. The number of generations would have directly corresponded to the how many generations it took in both problems to reach the optimal.

Below is a display of how the Python program works. The original image was provided along with the source code created by Derek.
Figure 5.3: The original input image and the output produced
Chapter 6

Evaluation

6.1 Overview

This chapter looks back at the project as a whole and the different aspects that led up to the development of the product and the development itself. The main part of the evaluation is to measure how successful the implementation of the minimum requirements outlined in the first chapter have been as well as the aim. Further evaluation is considered for the methodology chosen and the schedule.

6.2 Aim and Requirements Evaluation

The aim outlined in chapter one has not been completed. Whilst investigation has been carried out and research has been documented in chapter two, the simulations that were to be developed have not been completed. The plan of what each simulation was intended to do as well as how the simulation should be tested are discussed in chapters three and five respectively.

Requirement 1

The first requirement was to extend the existing software to incorporate the new simulations. This worked well and ultimately, the interfaces required for both of the proposed simulations were developed. While they do not have the full functionality that was planned, they have been designed following the appearance of the existing product, which was an intended. The interfaces have not, however been connected together so that all of the simulations are part of one overall environment but rather they are still three separate environments.
An additional welcome screen has been developed with the intention of using this to anchor the simulations together. This serves the purpose of allowing the user to select which simulation they want to use and once they have made their choice, the selected simulation will be shown.

**Requirement 2**
The second requirement was to develop the genetic algorithm to be used in the simulation environment. This was a very hard requirement to complete as attempts were made to build the GA from scratch. Through researching development of genetic algorithms, there are plenty of resources available, however incorporating them to fit with the environment was an additional challenge and ultimately, one that went beyond the scope of the project of this duration.

A genetic algorithm was still developed, as outlined in 4.1.5 and lays he ground work for a future project in this area to extend the genetic algorithm and incorporate it into the environment.

**Requirement 3**
The third requirement was to develop one of the discussed simulations and implement it. Towards the end of the project, much more progress was made in the development of not just one but both of the simulations meaning that one of the proposed extensions has also been achieved. The simulation that was intended to be completed at first was the Bee Foraging simulation. This choice was made through the course of the project as there were less parameters to be considered compared to the Cuckoo Search simulation, although some were still ignored for example, bees use the movement of the sun to make choices in how they travel and search for food. This was ignored as it was a complicated parameter to try and measure.

The Cuckoo Search simulation, although difficult to grasp the full extent of the parameters involved, still seemed to produce interesting results particularly relating to the application of the Levy flight process and how beneficial it was to the success of host birds gathering food as well as the cuckoo’s success of laying an egg in a host nest.

### 6.3 Evaluation of the product

Overall, the product that has been developed has fallen short of the original aim, however this does not mean it has been done in vain. One unexpected and pleasant outcome is that both simulations have been implemented rather than just one. Another aspect that fell short of original expectations was the development of the genetic algorithm as it is not integrated with the environment and instead operates as a stand alone program. Conversely, more focus was given to the Levy flight process which originally was overlooked but as ultimately been an interesting model to investigate.
6.4 Schedule Evaluation

To begin with, the project was following to the schedule quite well. The background research was straightforward and the gathering of resources was not difficult. Too much time was allocated to this however as a block of five weeks was provisionally given for this. This meant that while the reading was quite extensive, a slow start was made on the actual development of the product. When development began initial choices for design which failed to work meant that extra time had to be put into revising such choices which delayed development further. Once programming began, it was quite hard to stick to the outlined schedule as different tasks required different levels of effort and attention. As development continued, a trend was soon emerging whereby several features and classes were being worked on at once in attempt to get things working faster yet in hindsight, this was actually a hindrance and meant that development was slowed down again. Ultimately the programming took much more time than was intended.

The writing of the report was also scheduled to be carried out throughout the project however due to the difficulties encountered in the design and programming, time was taken away from this task meaning much of the report had to be written at the end of the project.

6.5 Methodology Evaluation

Earlier in the Introduction to this report (see section 1.1, a software methodology was selected to be followed during the development of the product. The methodology chosen was the Rapid Application Development approach. This methodology was chosen for its convenience in being applied to a project of this size as well as the benefits it provided with the prototyping aspect it brings. Having the structure of a methodology was very useful through the project as it helped to plan time better and gave clearly defined phases which made organisation of work clearer in terms of where it should be placed.

There were challenges in sticking to the methodology however, especially when it came to actually programming the product. The initial phase of analysis went very well as it did not take much time to become familiar with the existing environment and how it was working. There were some difficulties with the interface as text would appear and disappear intermittently but this did not have an adverse affect on being able to use the software. The planning was very clear cut and appropriate tasks were defined to be able to begin development.

The design phase is where difficulties started to occur and drastically affect the schedule that was
originally planned. At this point, development of the GUI began in the Netbeans IDE. This was a good starting point for developing the GUI as it worked almost like a storyboard, helping to plan where certain parts should be placed and provided a feeling of getting to know the GUI and how a user could think about navigating it. The problem with this though was that as more components were added, a lot of code was generated behind the scenes a lot of which was unnecessary and caused clutter. This code did not carry any of the functionality that the GUI would use and was purely for layout of the GUI. The code that was generated was also all produced within one class which removed the modularisation of code which was being planned.

While this was not implementation, it did have an adverse affect as the time spent revising the design took away time from the implementation and thus, not all of the intended features of the environment were included.

The construction phase led to further problems. Once the interface had been handled, development of a simple animation had to begin. This was very challenging in itself as this was tackled with several approaches with many of them not coming to fruition. An additional challenge at this stage was trying to keep in line with the original structure and organisation of sprites and animation in the existing product which meant that development of other approaches was limited as a conscious effort was made to not deviate too far from the design already in place.

Further problems occurred with organising the hierarchy of the GUI as well as the sprites and how they would be stored. Ultimately to fix some of the issues, the original ViewSprites class had to be revised to be compatible for the new simulations.

Getting a simple animation standing alone from the whole environment was not a hard task once it was developed however applying this in the program was difficult and took up much more time then was allocated for the task.

The testing phase was not fully explored. The benchmark tests that were intended to be used could not be incorporated because the program was not fully developed up to the standard that was planned thus obtaining the results from the simulation that were necessary to plug into the benchmark tests were not available. Parts of the interface were tested throughout the construction phase as the initial interface was very untidy and staggered development was necessary before it was completed.
Chapter 7

Discussion

7.1 Conclusions

At the outset of this project, the aim in mind was “To investigate the behaviour of particular animal species that have inspired the development of optimisation algorithms in a bid to improve the formulations of such algorithms or potentially create brand new algorithms”.

This aim has been a hard one to achieve and while it has not been completed to the standard that was anticipated at the beginning of the project, simple simulations have been developed.

Looking back at what was intended from the outset, it has now become clear that the two algorithms that have been investigated are stripped down to simplified formulations as they are specifically engineered with optimisation in mind thus ignoring some aspects of nature are appropriate for such formulations to work.

The bees algorithm of food foraging has been the most interesting one to investigate as it has given insight into the complexity of what seems like a simple task being performed by a simple creature when in actuality, bees are incredibly adept at performing many tasks and processing complicated information. As development of the bees simulation progressed, it became feasible to be able to produce the cuckoo search simulation also. This was also an interesting problem to investigate due to the method of breeding that cuckoos adopt. While the parameters involved were challenging to quantify, it was still a good and rewarding challenge.

The other difficulty was with the genetic algorithm but where this ultimately was not fully imple-
mented, it made way for considering the Levy flight process and giving more attention to the development and investigation of this. The Levy flight process itself, does have some complex mathematics behind it, some of which are documented by Yang [29], but building a simple model of movement that mimics some of what the process is intended to do has shown that it is a useful model and, especially for the bee foraging, has proved to be a successful one also.

In closing, with the original aim in mind, it is now clear that the two algorithms that have been considered here, are stripped down to simple formulations as they are designed for optimisation purposes, thus ignoring some aspects of what occurs in nature is appropriate for the formulations to work. Additionally, while not part of the original aim, investigating the Levy flight process has helped to explain to some degree how cuckoos fly around nests performing brood parasitism and has also shown a benefit for bee foraging in that the bees were mostly successful in every attempt at foraging that was observed.

What the investigation into the Levy flight process has also shown is that while the distance of travel is decided upon randomly, capping this to a small value seems to be the most beneficial in most cases, however this conclusion is only based on a small number of tests with a limited amount of time and it is possible and favourable to run the tests more thoroughly and take an average of several different experiments rather than taking the result of just one experiment.

7.2 Phases to be completed

7.2.1 Complete the Genetic Algorithm

The first phase left to be completed is the genetic algorithm itself. Building this from scratch was a difficult task and while there are several libraries available, to actually understand the full concepts and theory has been a steep learning curve. As already discussed, the genetic algorithm already operates simply on a binary string, generating a random initial string and performing the operations of: Selection, Crossover and Mutation to reach the target string.

The next step from this would then be to link the genetic algorithm to the two simulations. The original intention was to develop one genetic algorithm and modify this slightly for both problems however, due to the disparity between the two problems and the array of parameters in both of them, this meant this approach was not appropriate and so both problems would require their own separate GAs.

In the product that has been delivered, a class for the genetic algorithm has been defined and has the basic structure within. This class can be found in both the bee simulation and the cuckoo simulation.
7.2.2 Levy flight process

While a simple model of movement was achieved, it did not capture the full extent of the Levy flight process. This model was key to the work done by Yang [29] for the cuckoo search algorithm, but it was also a good model to be applied to the bee foraging algorithm also. The reasoning for this is that the actual method of movement of bees would be tough to encapsulate and beyond the scope of a project of this size and duration so applying the Levy flight process seemed a much better method and more easily definable.

From the model of movement that is already developed, the addition of the Levy flight model should not require much more development as the sprites are already moving in different directions in a random manner, all that would be required is to incorporate the method of turning at a 90 degree angle and rather than rapid random movements that are in place now, a more staggered method of movement that allows for cases where a bee or cuckoo may travel a greater distance before turning compared to others in the respective environments.

7.2.3 Apply GA to the simulation environment

The final major phase of development to be worked on would be incorporating the genetic algorithm within the simulations themselves. As stated previously, the vast array of parameters considered in each simulation means that to take just one GA and apply it to both would prove to be a fruitless task. Also the nature of each simulation with one being a foraging model and the other a breeding model means that the core process of the genetic algorithm will have to be modified for both in that the focus of the cuckoo simulation stays true to the core of a normal GA but, the GA for the bee foraging will have to be changed so that it orientated more for the actual process of foraging rather than breeding and foraging.

7.3 Further work

Given the current development of the project, there are several extensions that can be attempted to build on what has been developed as well as phases remaining from current development. Regarding the interface, what has been developed both through Matthew’s project and this one, the interface can be further refined to remove some bugs in its appearance. The interface for controlling the different parameters of the simulation can be temperamental and does not always update properly while being used, for example when a user interacts with a number slider the values of the slider occasionally disappear as well as the text beside them.

There are some more in depth extensions that can also be considered aside from the ones described above. Firstly, a different model of implementation can be considered that would be set apart
from the original sheepdog simulation as well as the genetic algorithm implementation used for the bees and cuckoo simulation. This would involve using Markov models, the reasoning for this being that it would keep in line with the different learning methods as Markov models are useful for reinforcement learning, particularly the Markov decision process. Another motivation for this model for implementation would lead to further building of the project to work as a general all purpose simulation environment for behaviours that happen in nature. Additionally, there are several different ways of implementing Markov models which can lead to much more extensions and room for development. Different Markov models include:

- Markov chain
- Hidden Markov Model
- Markov Decision process
- Partially observable Markov Decision process

A final additional extension can be to look at a brand new simulation. An idea that was originally envisioned as a standalone extension was to implement a simulation of fish schooling to investigate the benefits this provides when it comes to feeding and the interesting problem of predator avoidance.
Bibliography


[18] Peter Lifland Peter Bailis. The hive is hungry: Exploring bee colony search and foraging behaviour through agent-based simulation. 2010.


Appendix A

Personal Reflection

To say that this project has been a challenge would certainly be an understatement. There is no good description I can give to emphasize the challenge that this project does pose. I began with much ambition and an absolute determination to do my utmost to achieve the goals I wanted. Looking back, I have come across a lot of difficulties throughout the duration of this project that have pushed me to my limits when it comes to applying myself and applying techniques I have learnt through my time on this course.

Some challenges that I faced, much like many of my colleagues have been to deal with time management and planning, in fact there is no project that ever runs smoothly; no matter how well you plan, you can never be prepared for every eventuality and situation that may arise. This does not mean to say that planning should be ignored. I had rarely planned how to approach work in the past and through undertaking this project I feel I have taken away lessons in managing my work and motivating myself to work better.

Other lessons I learnt early on came with my background research. As I was reading, I made a critical mistake of not taking notes and kept very poor track of where I was obtaining information from. This had a quick knock on effect to my write up for the mid project report and delayed my original plan by approximately a week. Referencing was also a problem, I did all my reading without adding my references as I went so I found myself trawling through my writing later on which caused another delay. These mistakes were the most adverse challenges I faced early on and considering that the report itself is the main focus of this project, it was a very big error on my part to take such an approach and it cost me dearly.
My advice to students undertaking projects in the future regarding these aspects is to never underestimate the effort and the work required for writing the report. You cannot just sit down and attempt it in one sitting a week before the deadline as you need to plan what you want to say and, much as I have been doing, the project has to be written for an audience, it is not a personal monologue or diary. Also, a good point to remember is that the mid-project report is useful for testing the waters with your supervisor and assessor to see what they are looking for in your writing and I know from the feedback on my report that it helped me greatly for writing later on so that I could construct an appropriate report for the intended audience.

One other setback was that I told myself and as I indicated in my schedule that I would do writing throughout the entire project, including alongside implementation. You have to have a great amount of devotion to be able to carry out the two tasks alongside each other. If you know you can do it and that you can stick with it, then by all means it is a good approach but for me, I quickly realised I could not maintain a balance and as I got stuck on one aspect, I forced myself to go back to the other and I cost myself more time in this chop-and-change approach rather than staying focused to one task.

There are other challenges aside from planning and management that I faced through the duration of this project. Most of all, I did not appreciate how emotionally draining the project would be. To sit down everyday and for a time, doing the same work over and over, revising things that have already been done, trying several techniques at a time does become tedious, even if you’re enjoying what you are doing, there are times where you may feel that you are banging your head against a brick wall. This tedium then affected my overall productivity as the more I kept repeating things, the less I was getting out of what I was doing.

My biggest challenge in the project itself was using Java. In fairness, if I had used any programming language it would have been very demanding as I have never had to go so in depth or be as extensive with a language before, my only experience with project writing on this scale in the past was at college but a plus point of that was I had a timetable to follow whilst that project was being done. Not having a timetable meant that occasionally I had problems with making myself get up and go but other times, I realised my time was my own and I did not have anyone telling me what to do and when to do it by which in itself was quite liberating. This liberating feeling was not always a good thing though.

The one aspect of the project that kept me going perhaps more than any other was my friends. A lot of us would find ourselves in the lab together from time to time and occasionally collaborating but mostly we would take a break from our respective projects and we would talk about anything else. I felt this was good for me sometimes as the monotony of programming and writing was gradually
driving me mad as I rarely gave myself a proper break from doing work. For me, friends were a big help as without their insights and their feelings about the project as well the entire four month duration would have been a terribly lonely experience.

For anyone undertaking projects in the future, it is good for you to have an ‘off’ switch, even if you feel you have more to do and that you don’t want to stop until it’s done, you cannot go constantly, everyone needs some time to rest and it can be tremendously good to leave something alone for a while and come back and look at it with a fresher mindset, as I found when it came to coding and some of the writing.

I will not discourage anyone from doing the project as it is a good learning experience but the most important thing I would impart to anyone embarking on a project like this in future years is to ensure you pick a project that you love, do not pick something because you feel it will be useful or look good on a CV, pick it because it is what you want to do. I picked this project because I love Artificial Intelligence and through my time on the course, the most intriguing module I had the pleasure to undertake was the Biological and Bio-Inspired Computation module and having the opportunity to work on a project in the field has been, aside from problems, a very good experience and I know that I stuck to my passion and chose something that I knew I would, at least for the most part love to do.
Appendix B

External Resources

The external materials I used in this project include
Matthew Trafankowski’s project
Source code was supplemented by Dr. Brandon Bennett

GA example
A simple genetic algorithm, implemented in Java which inspired development of my genetic algorithm
can be found here: http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=3722lngWId=2

Image Reproduction GA
The source code for this program was created by Dr. Derek Magee for use in the A.I.20 Artificial Intelligence module
Appendix C

Ethical Issues

Throughout the course of my project there were no ethical issues that arose.
Appendix D

GUI
<table>
<thead>
<tr>
<th>Nesting Parameters</th>
<th>Eggs</th>
<th>Food Sources</th>
<th>Configurations</th>
<th>Sprites</th>
</tr>
</thead>
<tbody>
<tr>
<td>Species 1: Number of eggs generated in nest</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 egg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 eggs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 eggs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Species 2: Number of eggs generated in nest</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 egg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 eggs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 eggs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cuckoos: Number of eggs to be laid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 egg</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 eggs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Genetic Algorithm</th>
<th>Rate of Mutation</th>
<th>Rate of Crossover</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>