Visualisng Ant Colony Optimisation

Final Report for CS39440 Major Project

Author: Christopher Edwards (che16@aber.ac.uk)
Supervisor: Dr. Neil MacParthalain (ncm@aber.ac.uk)

April 9, 2015
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in
Computer Science (G400)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature .....

Date .....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature .....

Date .....

Acknowledgements

I would like to thank my supervisor Dr Neil MacParthalain who has provided incredible help throughout the projects development. I appreciate the time he has spent with me at various times which has allowed me to develop a greater understanding the underlying algorithm behaviours. I would also like to thank Neil Taylor who has been very informative in regards to what is expected from a major project. My appreciation also goes to everyone involved with the department of Computer Science at Aberystwyth University for proving the resources necessary for the completion of a successful project such as this.

My Thanks is also expressed to my fellow final year students, especially Thomas Keogh, for spending many hours in the Delphinium over the course of the projects development enabling the countless hours spent testing and debugging much more enjoyable. Finally I would like to thank my mother Diane, father Paul and brother Michael for continued support and motivation throughout my degree.

Abstract

Ant Colony Optimisation and its variations are commonly used swarm intelligence methods, however the underlying concepts can be difficult to comprehend for people who have recently come across the subject area. The majority of existing resources either inadequate visual representations or rely on the user having some prior knowledge about the underlying behaviours. The author of this project aims to create an application for deployment in educational environments allowing for a richer, more interactive experience in regards to the teaching of Ant Colony Optimisation methods. The author has set out to achieve a full visual representation of the algorithm's execution as well as providing an intuitive user interface allowing for user defined algorithm parameters and a choice of algorithm types and modifiers.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background & Objectives

This section should discuss your preparation for the project, including background reading, your analysis of the problem and the process or method you have followed to help structure your work. It is likely that you will reuse part of your outline project specification, but at this point in the project you should have more to talk about.

Note:

- All of the sections and text in this example are for illustration purposes. The main Chapters are a good starting point, but the content and actual sections that you include are likely to be different.
- Look at the document on the Structure of the Final Report for additional guidance.

1.1 Background

What was your background preparation for the project? What similar systems did you assess? What was your motivation and interest in this project?

1.2 Analysis

Taking into account the problem and what you learned from the background work, what was your analysis of the problem? How did your analysis help to decompose the problem into the main tasks that you would undertake? Were there alternative approaches? Why did you choose one approach compared to the alternatives?

There should be a clear statement of the objectives of the work, which you will evaluate at the end of the work.

In most cases, the agreed objectives or requirements will be the result of a compromise between what would ideally have been produced and what was felt to be possible in the time available. A discussion of the process of arriving at the final list is usually appropriate.

1.3 Process

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

Chapter 2

Design

You should concentrate on the more important aspects of the design. It is essential that an overview is presented before going into detail. As well as describing the design adopted it must also explain what other designs were considered and why they were rejected.

The design should describe what you expected to do, and might also explain areas that you had to revise after some investigation.

Typically, for an object-oriented design, the discussion will focus on the choice of objects and classes and the allocation of methods to classes. The use made of reusable components should be described and their source referenced. Particularly important decisions concerning data structures usually affect the architecture of a system and so should be described here.

How much material you include on detailed design and implementation will depend very much on the nature of the project. It should not be padded out. Think about the significant aspects of your system. For example, describe the design of the user interface if it is a critical aspect of your system, or provide detail about methods and data structures that are not trivial. Do not spend time on long lists of trivial items and repetitive descriptions. If in doubt about what is appropriate, speak to your supervisor.

You should also identify any support tools that you used. You should discuss your choice of implementation tools - programming language, compilers, database management system, program development environment, etc.

Some example sub-sections may be as follows, but the specific sections are for you to define.

2.1 Overall Architecture

2.2 Some detailed design

2.2.1 Even more detail

2.3 User Interface

2.4 Other relevant sections

Chapter 3

Implementation

The implementation should look at any issues you encountered as you tried to implement your design. During the work, you might have found that elements of your design were unnecessary or overly complex; perhaps third party libraries were available that simplified some of the functions that you intended to implement. If things were easier in some areas, then how did you adapt your project to take account of your findings?

It is more likely that things were more complex than you first thought. In particular, were there any problems or difficulties that you found during implementation that you had to address? Did such problems simply delay you or were they more significant?

You can conclude this section by reviewing the end of the implementation stage against the planned requirements.

Chapter 4

Testing

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on real users? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1 Overall Approach to Testing

4.2 Automated Testing

4.2.1 Unit Tests

4.2.2 User Interface Testing

4.2.3 Stress Testing

4.2.4 Other types of testing

4.3 Integration Testing

4.4 User Testing

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Initial Design Proposal

1.1 Introduction

1.1.1 Purpose

The purpose of this document is to give a detailed description for the design of the'Visualising Ant Colony Optimisation' application. This document will cover the proposed interface designs and interaction methods, choice of language and the underlying data structures and logical modules used in the application.

1.2 Language

The choice of implementation language for both the graphical user interface and the Ant Colony Optimisation algorithm is extremely important. The nature of the project suggests that an Object Orientated approach would best suit as the implementation language. One of the main reasons for this is because the manipulation and use of Objects in such languages allows for Object-based decomposition allowing for more logical modules system wide when compared to a non-Object Orientated approach. A non-Object Orientated approach may be more subject to functional decomposition (the application is split into modules grouping similar functions rather than representing separate Objects).

An Object Orientated approach has been identified as most suitable. Therefore there are two main languages which are at the forefront of the selection process. These two languages are C# (*C Sharp*) and Java. Both languages have the potential to achieve a high level of success when applied to the projects problem however; they both have different consequences depending on the environment and application in which they are used. C++ (*C plus plus*) is another popular Object-Orientated language. C++ has been discounted due to lack of language experience therefore using a more familiar language such as the two specified above (Java and C#) is far more appropriate in this instance.

One of the major factors in deciding on the implementation language is the suitability of the languages features when applied to the projects problem including any external resources or compatible libraries. Both Java and C# are very similar at an abstract level in terms of provided

features by default. Both include everything that would be necessary to implement the proposed design for this application. Both languages provide the ability for Objective decomposition and allow for polymorphic behaviours (multiple entities of different types using the same interface) which enables effective use of inheritance to allow multiple Agent variations (Ants in this case) to be supported easily.

C# has been created and is continually being developed by Microsoft and is focused around the .NET framework which is also a product of Microsoft. As C# is heavily Microsoft orientated its cross-platform capabilities are significantly reduced. The .NET framework(s) have only recently been open sourced so they lack full support on all platforms reducing the applications cross-platform reliability. The project is being developed with a focus on educational value; therefore cross-platform reliability is very important as maximisation of potential consumers is important (more people using the application implies more people are learning). A standard build will not be specified or assumed so there must be necessary measures in place to accommodate as many environments as possible. C# is heavily coupled with Windows based systems therefore; If C# were to be used there is a risk of alienating Macintosh and UNIX users. There are attempts to port .NET to other architectures (for example, Mono [3]) but the implementations of such approaches are not exact replicas of Microsoft .NET framework(s), therefore they cannot be relied upon. The use of Java would eliminate the cross-platform support issues as Java applications execute inside the Java Runtime Environment (JRE) which is available across most platforms and behaviours can be accurately modelled and predicted in the vast majority of cases.

Little differs between Java and C# in terms of feature presence (abstractly, how each language achieves each task is very different) thus, Java will be used for this project. The cross-platform constraints that come with the use of C# are not balanced by any necessary exclusive key features. As a result Java is the most appropriate language for the project, this all but ensures cross-platform reliability without sacrificing any important libraries or features.

1.3 Architecure

There have been considerations as to what key elements will be present in the composure of a suitable underlying architecture for the application. The architecture must accommodate both major elements of the application (graphical user interface and the Ant Colony Optimisation algorithm) in a manner that enables the best possible expansion/modification opportunities to accommodate any additional features or unforeseen changes. Selecting relevant Design Patterns will enable the above goals to become reality however; design patterns should be respectfully and must represent a general solution to a problem. The Overuse or misuse of such patterns can cause significant complexity issues through the system, this needs to be avoided.

1.3.1 Design and Architectural Patterns

1.3.1.1 Model-View-Controller

The Model-View-Controller (MVC) Architectural Pattern is designed to reduced coupling between system components, these are represented here as the user interface(s) (View) and the underlying data and its representation(s) (Model). The interaction(s) between the View and the Model "established using a subscribe and notify protocol" [1] (Controller). The Controller updates the View(s)

based on the model(s) current state or vice-versa however; The View(s) cannot directly communicate with the Model, the Controller must govern such interactions.

This project will make effective use of Model-View-Controller in order to produce an environment which is much easier to maintain and has little coupling between the Model and the View(s). This allows the Model(s) and/or View(s) to be substituted or modified in order allowing different representations of the current algorithm, or in fact different algorithms altogether.

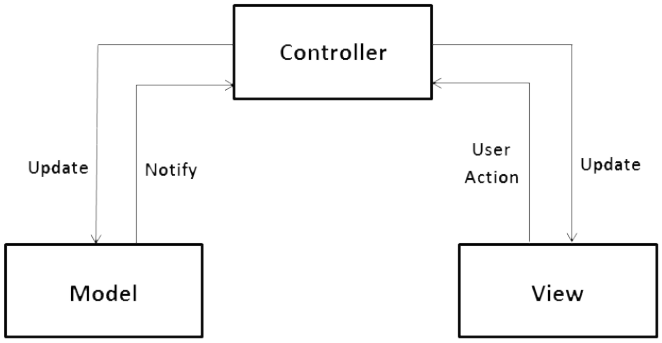

Figure A.1: Basic abstract overview of the Model-View-Controller pattern

- **Model** represents the underlying Ant Colony Optimisation algorithm. The Model also contains the required arithmetic functions and any additional operations required to execute the algorithm correctly.
- **View** represents the graphical user interface which will not only display the algorithms execution, but also enable the user(s) to modify the algorithms parameters.
- **Controller** represents the Observer required to enable interactions between the Model(s) and View(s).

1.3.2 Observer and Observable

The implementation of the Model-View-Controller design pattern is handled using the Observer and Observable interface provided as part of the default Java language specification. The general premise is that the Model will have an Observer which will have an update interface allowing it to receive signals from what it is observing (Model). The Model will implement the Observable interface which will enable it to send update signals to the Observer through the notifyObservers() method. This enables the Models dependencies to be updated as the Model itself changes ensuring the correct state of the Model is captured in the Views.

As figure A.2 demonstrates the View will Observe the Model and will wait for an update signal sent by the Model's notifyObserver() method. The Model can have multiple Observers using this pattern so there is the potential for future modification or enhancement without having to rework the existing system should the needs for extra views be needed.

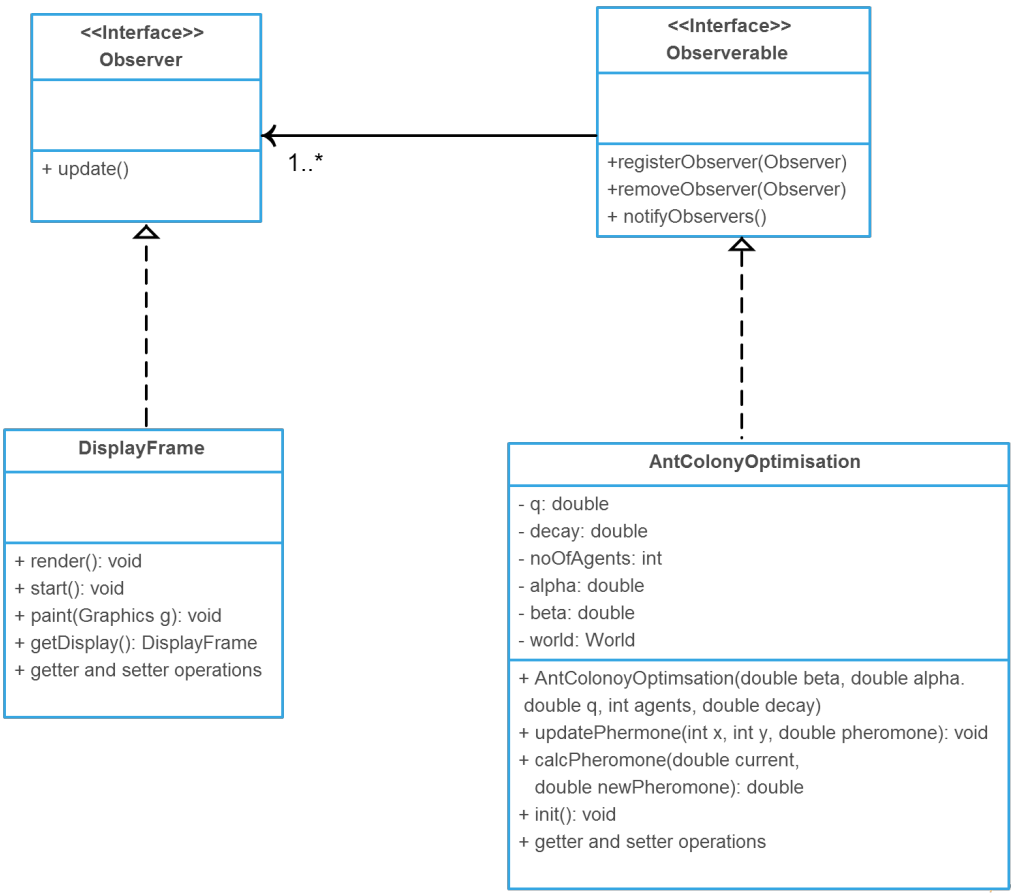

Figure A.2: Proposed implementation of the Observer and Observable Design pattern

1.3.2.1 Singleton

In order to maintain simplicity throughout the application the Singleton design pattern will implemented for key Objects where one and only one instance of an Object must exist. The Singleton pattern prevents multiple instantiations of specific Object(s) as the Object itself is solely responsible for tracking the currently instantiated instance of itself [2].

The application will consist of a graphical user interface which in turn, will be composed of several different components. Such components must only be instantiated once in order to ensure correct interactions are performed. Without the presence of the Singleton pattern there exists the possibility of multiple instances of such components which could potentially cause unforeseen complications and undefined behaviours.

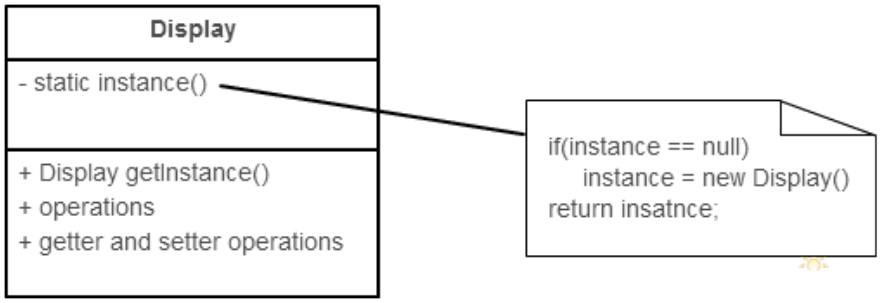

Figure A.3: Proposed high-level implementation of the Singleton pattern demonstrating how the sole instance of the graphical user interface will be tracked.

1.3.3 Structure

Adhering to the concepts of the patterns described in sections 1.3.1.1 and 1.3.2.1 the following Class Diagram shows proposed application structure. This is not concrete and could potentially change throughout development, the Class Diagram in question is represented below by figure A.4 and is represented using standard UML notation.

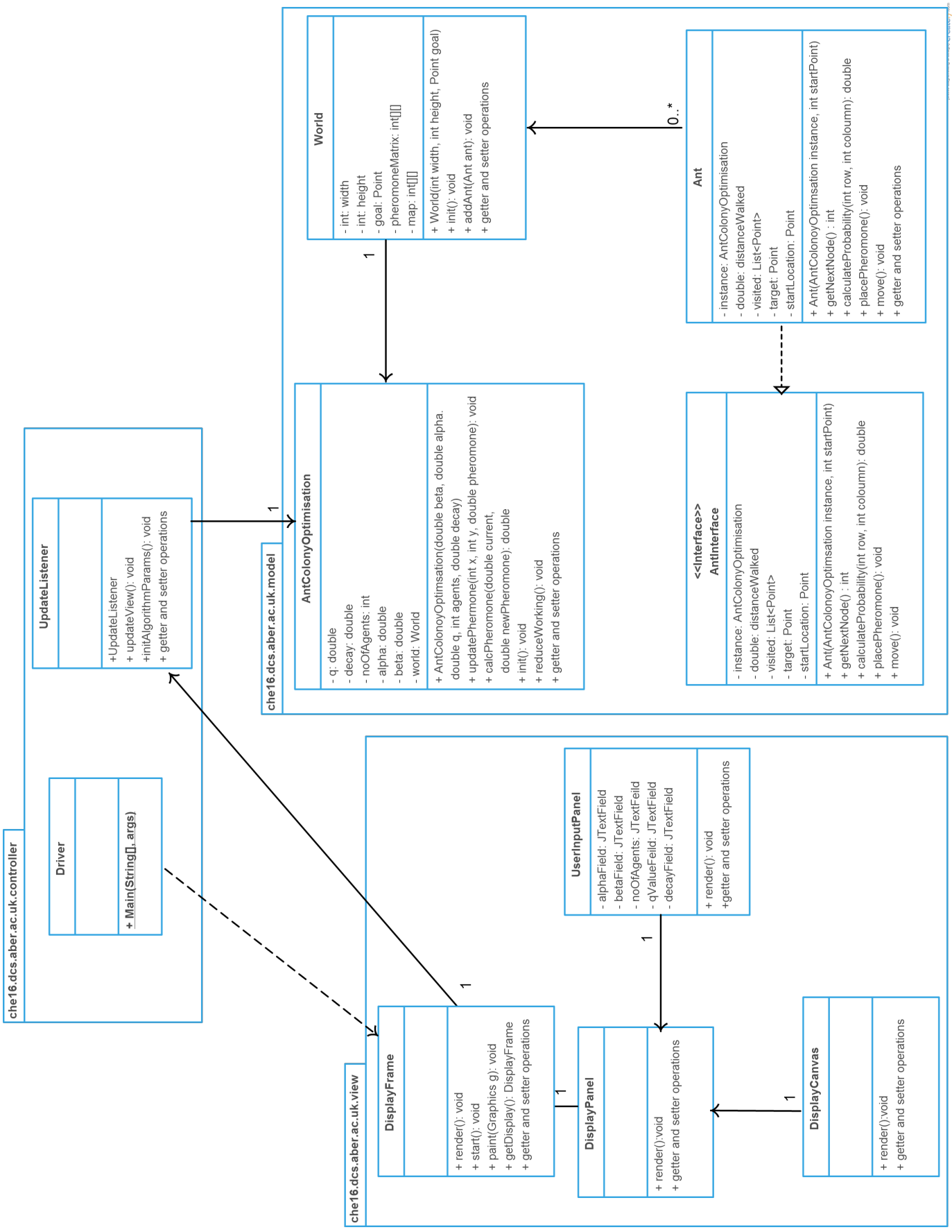

Figure A.4: Initial Class Diagram representing the main modules, packages and system interactions using UML notation.

1.3.3.1 View Package

The $che16.dcs.aber.ac.uk.view$ package contains the graphical user interface components. The main concept behind this is the use of nested components such as JPanel, JTextFields and the like. These will be contained inside top level JFrame. This allows modification of each component in isolation without impacting the other components behaviours and/or elements. This is the first module of the application which is initialised, and is done so directly from the $main$ method. This package has no direct knowledge of any package members mentioned in 1.3.3.2. Instead the $che16.dcs.aber.ac.uk.controller$ package members handle the communication(s) between the model(s) and view(s), adhering to the principles discussed in section 1.3.1.1.

1.3.3.2 Model Package

The che16.dcs.aber.ac.uk.model package contains the necessary elements and attributes required in order to accurately represent the algorithms state(s) and ensure correct algorithm execution. This package has no directl knowledge of any package members mentioned in 1.3.3.1. Instead the controller package members handle the communication(s) between these the model(s) and view(s), adhering to the principles discussed in section 1.3.1.1.

1.3.3.3 Controller Package

The initial concept for the controller is basic and will grow in complexity during the development lifecycle as more and more control based mechanisms will become prevalent. This initial concept is a simple Observer which notifies the view(s) should the model change in a significant way; the inverse of this is also true.

1.4 Interface Design

1.4.1 Main User Interface

The interface design must accommodate all the necessary elements required to allow user defined values for each of the algorithm parameters. The interface must also be able to visually represent the current algorithms state including the locations of the agents and nodes without impacting on the usability of the application (the interface must not freeze or be negatively affected by the algorithms execution).

The interface will use a very neutral colour scheme which will maximise the usability and reduce the risk of complications which may arise from users being subject to difficulties understanding or identifying certain colours. Every option for the user will be textually defined and will not rely on any sound or visual prompts in order to user. As a result in addition to the above, users with visual impairments will still be able to use the application as intended.

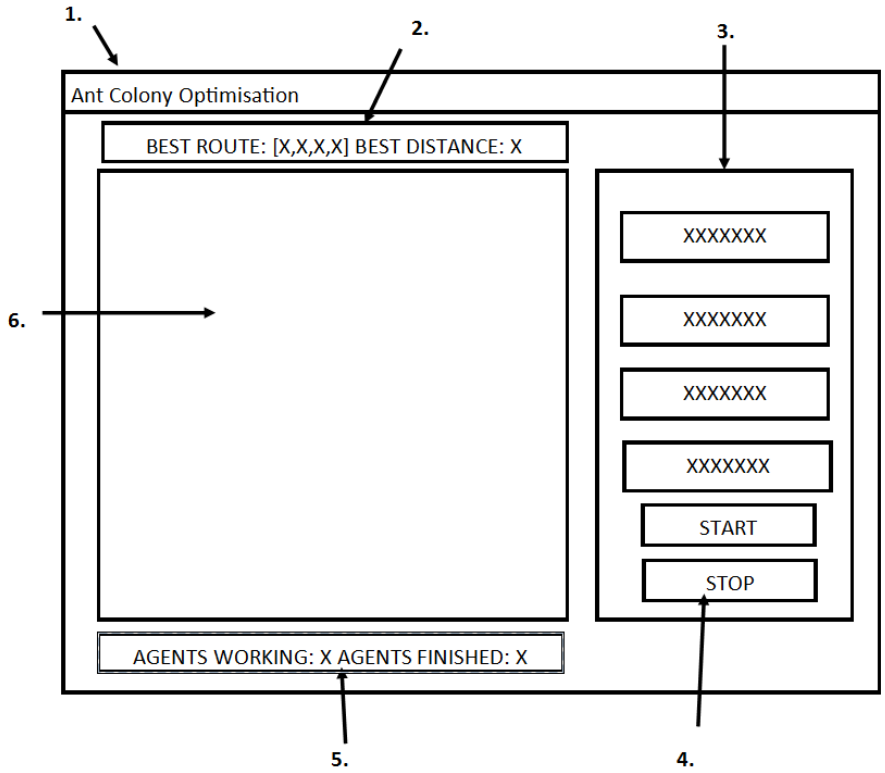

Figure A.5: Proposed design for the graphical user interface, showing the key elements in their planned locations.

1. in figure A.7 refers to the containing frame which will house the reset of the interface elements. This frame will not be resizable preventing complications such as the dynamic resizing of the interface having an impact on the observable range of the world. **2.** in figure A.7 refers to a text view containing information relevant to the algorithms current state of execution. This text view will contain the current best distance as well as the order of nodes visited to traverse the current best path. This will only be populated during the algorithms execution and only if the best path has been initialised. **3.** in figure A.7 refers to container which houses the interactive elements relating to the modification of the algorithms parameters. This will consist of several labels and text fields which can be modified informing the user of what they will be modifying as well as providing suitable error messages if the users enters an incorrect value for any of the parameters. **4.** in figure A.7 refers to the start and stop buttons. These are here to conform to the expectation that a user will expect some clear way to start and stop the application at their free choice, this is the simplest way to do this, and requires no hidden menus or hidden key bindings. **5.** in figure A.7 refers to a text view containing information relevant to the algorithms current state of execution. This text view will contain the number of agents currently working (which means the number of agents who haven't met their own stop conditions) as well as displaying the number of agents which are finished. **6.**in figure A.7 refers to the main canvas area which will display the algorithms current state of execution to the user. The contents of this will reflect the user defined values (elements in: **3.** in figure A.7) as well as representing each agents current location, their movements between nodes and also the modelling of pheromone deposit and decay will be present in this canvas.

1.4.2 Error Message Feedback

As the Algorithm parameters will be user defined using the interface proposed in section 1.4.1, figure A.7 there must be measures in place to catch and inform the user of any illegal values. Not only must the user be told they have inputted illegal values the illegal parameter value will be identified and a range of legal values will be displayed to the user.

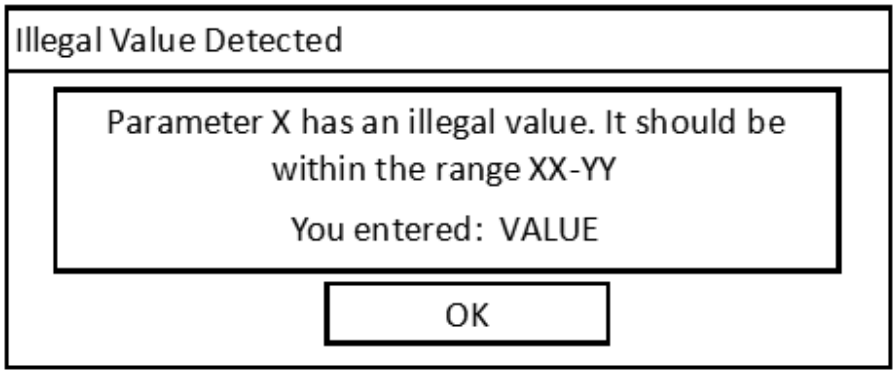

Figure A.6: Proposed design for the illegal parameter error message displayed to the user

As represented in figure A.6 the error message will contain all the information a user needs to identify the problem with their data input. The X value in figure A.6 represents which parameter has the illegal data input (i.e. Alpha, Beta). Range XX-YY represents the legal values for said parameter, and VALUE represents the value the user has specified for this parameter. The combination of these will provide the user with the knowledge of why this error message is being displayed and how they can resolve their issue.

This kind of error message will be composed using the JOptionPane and JDialog interfaces provided as default by the Java language specification. These views are very customisable and the styling will be handled by the languages underlying protocols which will reduce the codes complexity.

1.5 Algorithm

There are several adaptations of the Ant Colony Optimisation algorithm. Initially the project contain an implementation of Ant Colony Optimisation in its simplest form, without the presence of any enhancements such as using *Elitist Agents* or similar. Once a working implementation is in place the next step will be to adapt the algorithm in various ways to further aid the teaching potential of the project.

The general premise is that each Agent (Ant) embarks and a pseudo random walk through the state space. The Agents movements are influenced by pheromone deposits placed on edges between vertices. This pheromone is deposited by other Agents in accordance with the equations stated in section 1.5.2.2. The Agent's next move is influenced by the result of the equation stated in section 1.5.2.1. However; there is still the probability of the Agent moving to a less attractive point so the Agent does not always travel to the strongest pheromone concentration.

Overtime the pheromone deposit concentration on $edge_{xy}$ is directly proportional to the quality

of the candidate solution, and ultimately the ants will converge to find the shorted route between two or more points.

There are several algorithm requirements;

- **Suitable problem representation** the World and Agents must be represented in a suitable and logical manner allowing the algorithm to execute as expected.
- **Pheromone manipulation metrics** there must exists adequate ways to access the pheromone matrix as well as manipulate (deposit/remove) the concentration of pheromone on a given edge in order to model decay and deposits.
- **Probabilistic movement functions** there must exist functions that calculate the probability of the Agent moving to a specific vertex. This is based on the pheromone concentration and the Agents location (see section 1.5.2.1).

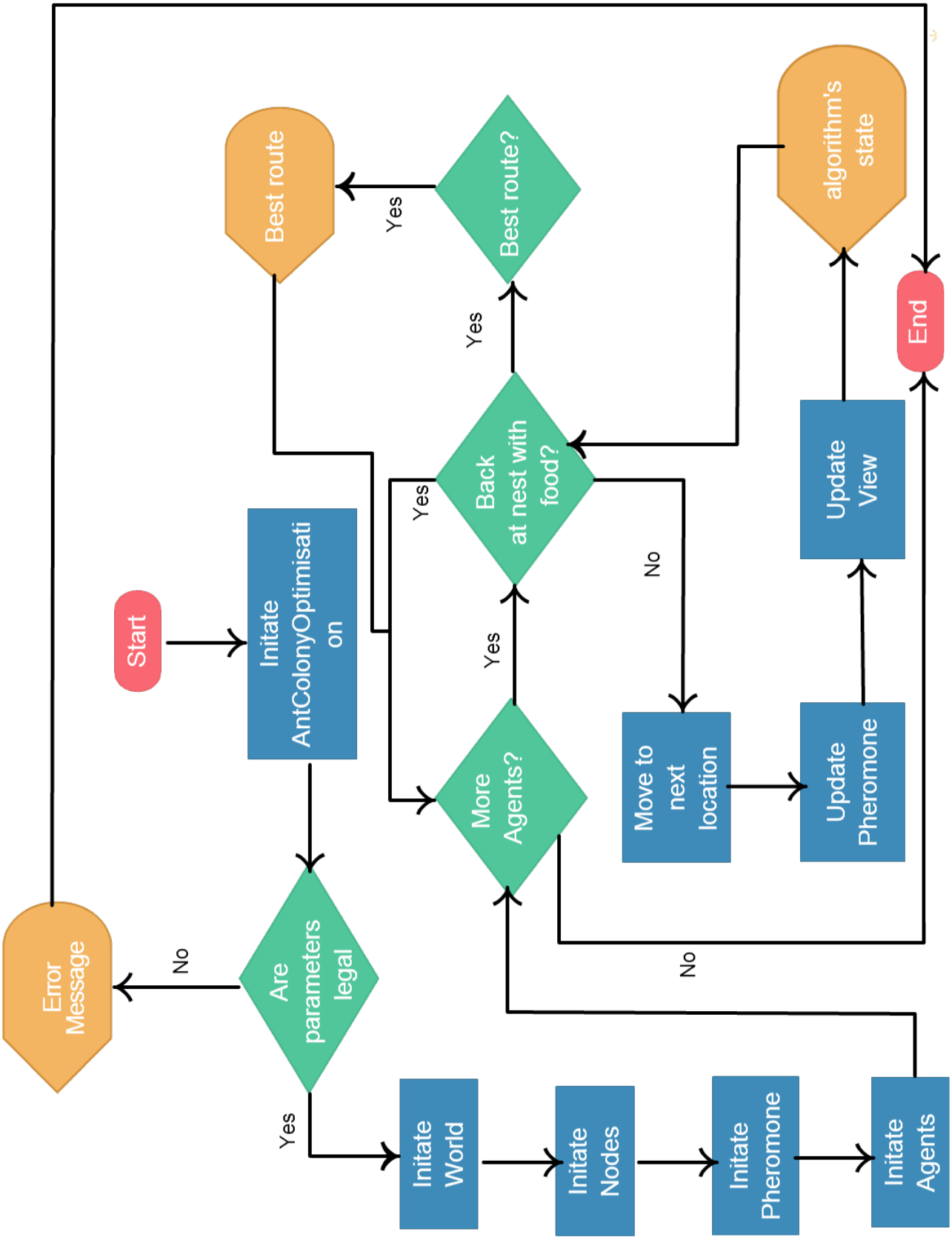

Figure A.7: High level flow digram representing the psudeo code for Algorithm 1, section 1.5.1

1.5.1 Pseudo-code

Algorithm 1 Pseudo-code for Ant Colony Optimisation

```
1: Initiate AntColonyOptimisation with defined parameters
2: if !parameters are legal then
3:     Dispaly error message to user
4:     return
5: Initiate World with algorithm parameters
6: Initiate Nodes and graph
7: Initiate pheromone values
8: Initiate Agents
9: while !all agents finished do
10:    for all Agents do
11:        while !back at nest with food do
12:            Calculate next move using probabilistic function
13:            Add moved point to Agent's memory
14:            Calculate and deposit pheromone on the path
15:            Update the View
16:        end while
17:    end for
18: if local best solution < global best solution then
19:    globalbest = local best solution
20: end if
21: end while
22: output global best solution
```

Above is a somewhat simplified pseudo-code representation of the proposed Ant Colony Optimisation algorithm. The mathematical formulae required to achieve steps *7* and *10* are shown in section 1.5.2. The final algorithm may differ from the above depending on any additional feature present in the final release.

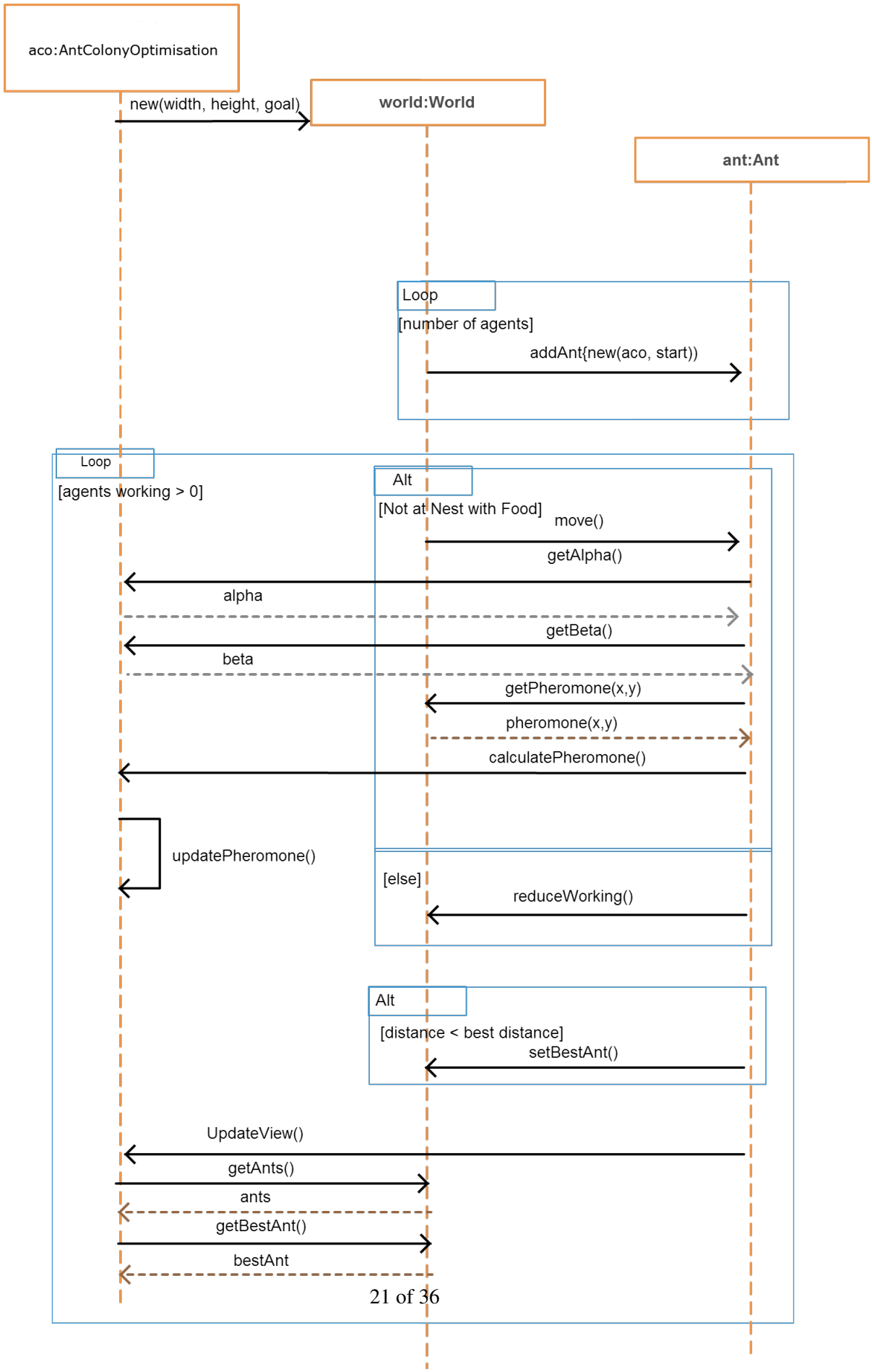

Figure A.8: Sequence diagram representing Algorithm 1, section 1.5.1

The sequence diagram shown in figure A.8 shows the high level interaction between the proposed main Model Classes during the algorithms execution. The Ants are in constant communication with the AntColonyOptimisation and World instances during their movement through the graph to ensure the correct pheromone values are being parsed during the movement process. The View is also updated through the updateView() method in the AntColonyOptimisation instance which requires data about the current Ants and the current best Ant. This data is stored in and returned from the World instance.

1.5.2 Metrics

1.5.2.1 Probabilistic function

$$p_{xy}^{k} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})} \tag{1}$$

Figure A.9: Algebraic model of the probabilistic function used to calculate next move for any Agent [7]

The probability that an Agent moves to vertex xy is described using the above. p is the probability for any Agent k to move through vertex xy. τ is the amount of pheromone deposited on vertex xy, which is raised to the exponent α. α is an heuristic value representing how greedy the algorithm is in its path finding [5]. The result for τ_{xy}^{α} is then multiplied by the edge xy's evaluation(η). Generally η will be represented using $\frac{1}{Euclidean\ distance_{xy}}$ [5] [4]. This will then be raised to the exponent β which like α is an heuristic parameter however β describes the Agents path finding speed.$\sum(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})$ is the sum of all possible solutions.

Algorithm 2 Pseudo-code for Probabilistic function - Each Agent - figure A.9

1: read the $pheromone$ level for vertex xy
2: raise the value from 1: to the exponent α
3: Multiply the result of 2: by $(inverted\ distance_{xy})^{\beta}$
4: initiate a temporary double $cloumnTotal$
5: **for all** visted vertex **do**
6: $columnToal$ += (read the $pheromone$ level for vertex $xy)^{\alpha} \times (inverted\ distance_{xy})^{\beta}$
7: **end for**
8: divide the result of $3:$ by the result of $5:-7:$

1.5.2.2 Pheromone deposit

$$p_{xy}^{k} = (1-\rho)\tau_{xy}^{k} + \Delta\tau_{xy}^{k} \tag{2}$$

Figure A.10: Algebraic model of the pheromone deposit function used to calculate the correct values for the pheromone matrix [8]

τ represents the pheromone deposit for an edge xy by Agent k [5]. ρ is a value between $0 - 1$ which represents the decay rate $decay$. $1 - \rho$ is multiplied by the existing amount of pheromone at $edge_{xy}$ to correctly account for decaying trails. The new amount of pheromone is then added using the equation from figure A.11.

$$\Delta\tau_{xy}^{k} = \begin{cases} Q/L_k & \text{if Agent } k \text{ uses curve } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Figure A.11: Algebraic model of the function used to calculate the how much new pheromone is deposited at xy [6]. Pheromone is only updated if the Agent k has visited point xy.

figure A.11 represents the new amount of pheromone to be added to the existing concentration at $edge_{xy}$. This can be read as change in τ $(\Delta\tau)$. Q is simply another heuristic parameter which is divided by the distance $agentk$ travelled to get to $edge_{xy}$. If the result of this is ≤ 0 return 0. This ensures that new $pheromone$ is only added to the existing concentration at $edge_{xy}$ is used by $agentk$ in its tour.

Algorithm 3 Pseudo-code for Pheromone function - figures A.10, A.11

1: **if** $pheromoneDeposit = Q\ value/totalDistanceWalked$ ¡ 0 **then**
2: $pheromoneDeposit = 0$
3: $pheromone_{xy}$ = (1 - $algorithm\ \ decay\ \ rate$) $\times$ $currentPheromone_{xy}$ + $pheromoneDeposit$
4: **if** $pheromone_{xy} \geq 0$ **then**
5: $pheromoneMatrix_{xy} = pheromone_{xy}$
6: **else**
7: $pheromoneMatrix_{xy} = 0$
8: **end if**

1.6 Representation

1.6.1 World

The World Class is used to model the graph that the Ants will traverse during the algorithms execution. The graph will be represented as a two-dimensional Array, with each element in said array representing a Node in the graph itself. The two dimensional array will be composed of integer values with each element containing an integer value representing the terrain type at this index, this will also be used to set the nest and food locations.

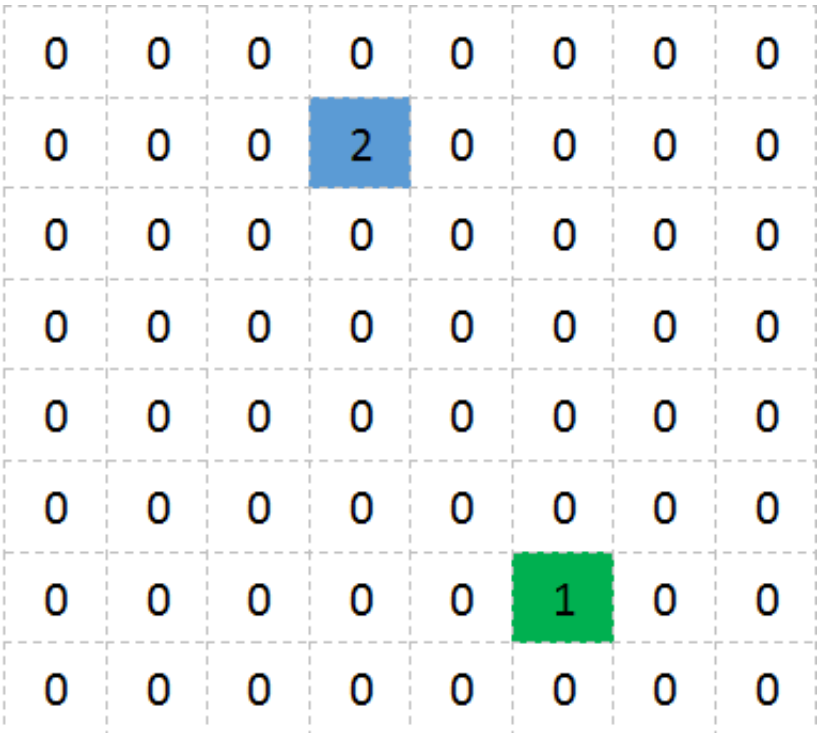

Figure A.12: Example World representation using a two-dimensional Integer Array

Figure A.12 shows how the proposed graph representation would be implement. A '0' value represents normal terrain where the Ants can move to, '1' represents the nest location where the Ants will start and '2' represents the food location which is the initial target for the Ants. This is a simplistic way to model an environment for the Ants whilst also allowing a multitude of terrain options without having to modify the way the graph is stored.

1.6.2 Pheromone

The pheromone must be modelled in a way which is both easily accessible and modifiable. The data structure used to store the pheromone must also relate to the graph mentioned in 1.6.1 to allow for logical mapping between the representation of the Ants environment and the pheromone associated with each Node in the graph. The pheromone will be stored as a two-dimensional Array of Doubles. Each element in the Array will represent the pheromone concentration for the corresponding Node in the graph for example, the double value at pheromone[x][y] will represent the pheromone concentration on edge [x][y] in the graph, thus there is a logical link between the two representations.

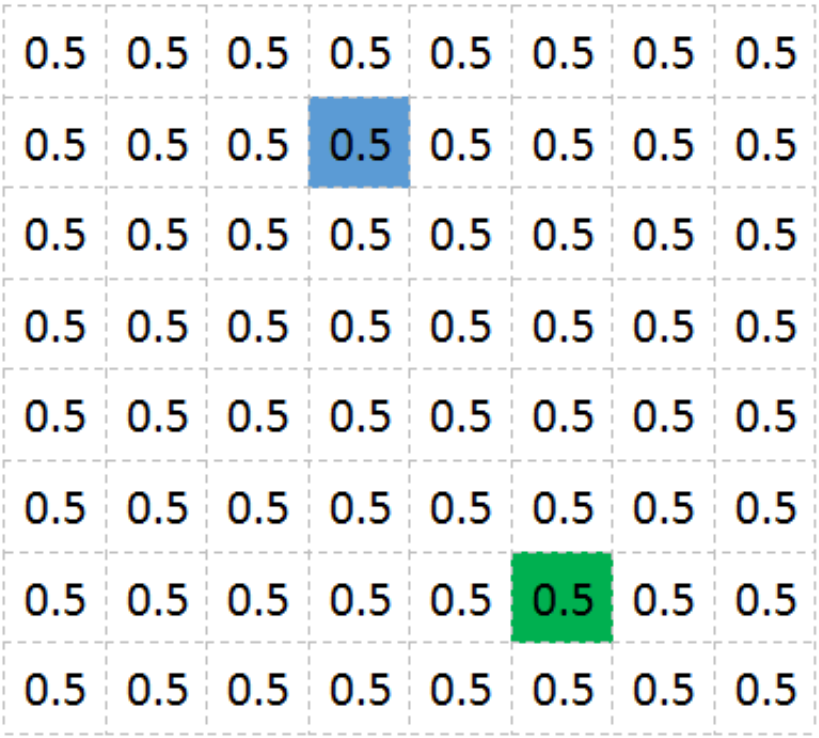

Figure A.13: Example Pheromone representation using a two-dimensional Double Array with default initial values

Figure A.13 demonstrates how a two-dimensional Double array can be effectively used to model the pheromone values for every Node in the graph. The values in this figure are default initial values which will be user defined when the application is executed. As the Ants moves through the graph the pheromone values at each index will correctly model the pheromone operations mentioned in section 1.5.2.2. As the values are stored in an array accessing or modifying the values is extremely simple and involves a simple getter or setter operation.

1.6.3 Visualisation

The data structures described in sections 1.6.1 and 1.6.2 must be visualised to the user in a manner that anybody can understand. Given that the graph will be stored as a two-dimension array it is perfectly logical to display this graph to the user in a grid format similar to that shown in figure A.12. The challenge with the visualisation process is visualising the pheromone deposit and decay operations as well as displaying the Ants moving between Nodes.

Given that the pheromone is stored in a two-dimensional array of doubles and the fact that accessing these values is extremely simple the value at each index can be used to directly influence how the user sees the pheromone. If each cell in the grid is coloured, and this colour's opacity is directly representational of the value at the corresponding index in the pheromone matrix then it becomes an accurate way of displaying the values of the pheromone to the user, which also allows the decay and deposit operations to be shown.

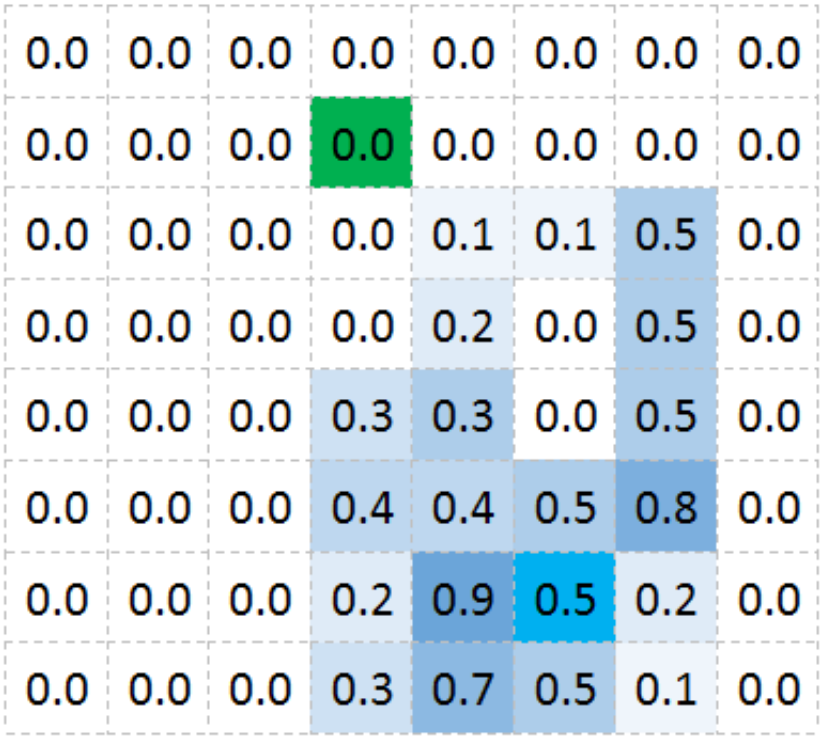

Figure A.14: Example Pheromone visualisation

Figure A.14 shows how the pheromone value for a given [x,y] co-ordinate has a direct influence on the opacity of the cells colour. In said figure the pheromone value at each index is multiplied by 100 to convert it to a percentage, which then becomes the opacity value for the given Node. However, if the pheromone values for each Node become extremely small during execution there must be a measure in place to covert the small values back into percentages using a larger scaling factor than 100.

Appendix B

Requirements Specification

2.1 Introduction

2.1.1 Purpose

The purpose of this document is to give a detailed description of the 'Visualising Ant Colony Optimisation' application. This document will cover the interface interactions and methods as well as providing definitions to important terminology. The document is primaraly intended to be used as a reference point for the initial stages of development.

2.1.2 Scope

The 'Visualising Ant Colony Optimisation' application is a desktop application which is designed to demonstrate the behaviour of an underlying ACO algorithm given various algorithm parameters. These parameters will be defined by the user and can be modified at their convenience. The algorithms behaviour given these parameters will be visible and the users can make a clear assessment about how each of the parameters impacts the algorithms performance.

During the background research into this subject area there does not seem to be too many applications which offer ACO visualisations and there are even less which provide a 'friendly' environment which is simple and intuitive to use regardless of the users background knowledge in regards to the subject area.

The software will be deployed in an educational environment, and aims to provide a means for teaching ACO to students as part of an Artificial Intelligence course or for independent use as a self-learning exercise. As a result the software must cater for the majority of user groups to maximise its effectiveness. This means the software must be accessible on all major platforms and perform equally well on said platforms.

Term	Definition
ACO	Ant Colony Optimisation
user	Anybody who is using or wishes to use the software
user group	A collective group of users representing different user needs.
interface	A graphical user interface
standalone	Operates independently of other hardware or software
agent	The entities which will be traversing the graph

Table B.1: Definitions for the keys terms used throughout this document

2.1.3 Definitions

2.2 Overview

This section will give an overview of the proposed application. This section will also expand on the expected user groups and functionality required by said groups. The constraints

2.2.1 Product Descriptive

The software application will be standalone and does not need to communicate with another system or application, because of this there is no need for any form of network connection to be present in order to use the application to its maximum potential.

The application will communicate with the Operating System on the host machine in order to enable the save and load functionalities through simple file input and output. However the users access to the host machines file system will be restricted by the fact that the saving and loading will be restricted to the users home directory preventing the overwriting of important documents.

The application itself will not take up too many system resources even if a large problem is being handled. This allows the users on a system or network to run the application without it impacting the performance of other services. Given that the target audience is educational establishments this is especially important as many teaching fellows have multiple applications running during a lesson or lecture and a negative impact on their system could reduce the amount of information taught during said session.

2.2.2 Product functionality

The users will be able to view a world which represents the Agents and the nodes in the graph otherwise known as the world. The state of this world will be directly related to the algorithm parameters specified by the users using the interface provided. There are several parameters which can be modified by the user, each of which will have a different impact of the state of the word and the algorithms behaviour.

The parameters which can be modified will be clearly labelled and will be obviously editable. As these parameters will be user defined there will be strict error checking measures in place to catch any illegal values before they can cause problems for the algorithm, and in addition each parameter will have a range of legal limits applied to them. This will prevent the users from

entering values of the incorrect type (String when the systems needs a double) and will also prevent values from outside of the specified range being accepted. When a complication or error arises there will be a simple error message presented to the user informing them of both the error and why it occurred which should enable the user to resolve what they did incorrectly.

2.2.3 User Groups and Characteristics

There will be three main user groups associated with this application. Each of these user groups will interact with the application in a different manner but the main purpose and result will remain the same.

Teachers/Teaching fellows will use the application with the underlying knowledge already in hand. They will be mainly be using the applications to visually portray ideas and will have expectations in regards to what to expect for a solution and will have some idea how the parameters impact the final result.

Students will use the application with some background knowledge of the underlying concepts but will still use the application in an experimental manner and may have little expectations or understanding of how changing certain parameters impacts the final result.

People new to the subject area will use the application with potentially no idea about the underlying concepts. There will be measures in place to explain the underlying metrics and give an insight into what the application is actually doing. Given that they have less knowledge of this subject area that the other two user groups mentioned above, they will still be able to achieve the same results and levels of functionality. The application will cater for all users regardless of prior knowledge.

2.2.4 Constraints

As the application is standalone is reduces the amount of constraints which it becomes subject to. The main constraint which the application is associated with is the dimensions of the users display. The interface has to house a lot of elements in order to produce a simple and effective environment, thus it takes up quite a lot of screen real estate. However in modern times the amount of space required for the application to perform as expected is far from unreasonable and the application will be developed with this in mind.

The algorithms execution time directly proportional to the user defined parameters, more specifically the number of agents and the number of nodes in the graph. The more agents and nodes the larger the execution time and resource requirements will be. The application will be developed with this in mind as there will be a constraint on how much memory and system resources the application should use. There is no expectation on the user to have a superfast high end machine therefore the application will be designed to accommodate a standard machine for these modern times and correct limits will be placed on these user parameters.

The application does write the host systems file store so there must be adequate room to do so, however the files that will be written are simple text files which will not take up a lot of room

on the host machines disk. Depending on the users machine this could still be a constraint. The responsibility and handling for this will belong to the users machine.

2.2.5 Assumptions

It will be assumed that the user will have the correct drivers installed and their machine will be able to handle the algorithms execution. The applications algorithms are not too resource intensive, therefore this is a reasonable assumption given the modern era and the advancement of computer technology.

It will be assumed that users will meet the minimum display requirements thus no dynamic resizing of the interface based on the users display dimensions will be performed. This significantly reduces complexity.

Another assumption is that every user will have some experience of using similar software and the interface will be familiar and therefore will be easy to use and navigate. The interface will use traditional methods such as simple buttons, text boxes and drop down menus to provide the user access to certain functionality.

2.3 Specific Requirements

This section contains the function requirements of the software as well as giving details about the different interfaces.

2.3.1 User Interface

There will be one main user interface for this application. This interface will contain a display area which is where the algorithms current state of execution will be represented visually to the user. There will also be an area which will house the text boxes which will be used by the user to interact with the algorithm and modify the parameters.

The display area will be simple and will be clearly identifiable. The text boxes responsible for handling user inputs will be clearly labelled so the user knows exactly what parameter they will be modifing. The text boxes will be obviously editable and the user will associate the look and feel of these text boxes with the fact that their contents can be modified.

The display will represent everything about the algorithms current state. This will display all of the graphs Nodes and all of the Agents at their current Node. The display area will also show the current pheromone levels for each connecting edge for a given node, this will be done in a way that is clear and understandable by all of the user groups mentioned in section 2.2.3.

There will also be a textual representation of the current best agent. This will display the best route and the distance of the best route and will update as the global best is updated. There will also be textual representations of how many agents are currently working in addition to how many agents have finished.

2.3.2 Hardware Interface

The application does not require any specific hardware or host environment as it will be fully cross-platform compliant there are no direct hardware interfaces. There is no network use in this application thus there is no need to communicate with network adapters or anything of similar nature. The system interactions between this application and the host's Operating System file system will be delegated to the Operating System itself.

2.3.3 Functional Requirements

ID: FR1
Title: Launch the Application
Description: Regardless of the users host environment the application should be able to be launched by the user using an executable .jar file.
Dependencies: None

ID: FR2
Title: Generate a World
Description: The user must be able to randomly generate a world for the algorithm to be executed on.
Dependencies: FR1

ID: FR3
Title: Visualise a World
Description: The user must be able to visualise the world and its parameters including the number of agents, the agent locations and the number of Nodes.
Dependencies: FR1, FR2

ID: FR4
Title: Provide a means to modify parameters
Description: The application must provide simple ways to modify the algorithms parameters.
Dependencies: FR1

ID: FR5
Title: Generate a World with specified values
Description: The user must be able to generate a world for the algorithm to be executed on. This World with have user defined properties such as the number of Nodes and Agents.
Dependencies: FR1, FR4, F10

ID: FR6
Title: Visualise the Pheromone
Description: Every edge in the graph will have its own pheromone value. This must be visually displayed to the user and correctly model the pheromone deposit and decay operations.
Dependencies: FR1, FR2, FR3, FR13

ID: FR7
Title: Visualise the Agents movement
Description: As the algorithm is executing the Agents will move through the graph. The movement of these Agents must be displayed to the user in a logical manner.
Dependencies: FR1, FR2, FR3, FR8

ID: FR8
Title: Start the Algorithm's execution
Description: There must be a simple way for the user to start the algorithms execution.
Dependencies: FR1

ID: FR9
Title: Stop the Algorithm's execution
Description: There must be a simple way for the user to stop the algorithms execution anytime the user wishes to.
Dependencies: FR1

ID: FR10
Title: Validate parameters
Description: As the users will be able to define thier own parameter values there must be correct measures in place to ensure the values entered are legal. If they are indeed illegal then suitable error messages will be displayed.
Dependencies: FR1

ID: FR11
Title: Display the best path
Description: As the algorithm is performing its task there must be a way to display the current best result to the user.
Dependencies: FR1, FR2, FR3, FR5, FR7, FR8, FR12

ID: FR12
Title: Agents must be able to move between Nodes
Description: In order for algorithm to perform as expected there must be a way for each Agent to move between Nodes in the graph.
Dependencies: FR1, FR2, FR3

ID: FR13
Title: Model Pheromone operations
Description: There must exist a way for the algorithm to correctly deposit and decay pheromones on graph edges adhering to specific formulae.
Dependencies: FR1, FR2, FR3, FR11

ID: FR14
Title: Load Configuration from a file
Description: There must exist a way for the users to load a pre-existing configuration from a file

of their choosing. This allows the algorithm to be executed on the same problem multiple times.
Dependencies: FR1, FR2, FR3

ID: FR15
Title: Save Configuration to a file
Description: There must exist a way for the users to save the current configuration to a file of their choosing. This allows the algorithm to be executed on the same problem multiple times.
Dependencies: FR1, FR2, FR3

ID: FR16
Title: Exit the application
Description: The user must be able to exit the application completely killing the process and freeing system resources.
Dependencies: FR1

2.3.4 Requirement Evaluation

Each of the functional requirements mentioned in section 2.3.3 differ in thier levels of importance. The dependencies field for each requirement donates which requirements must be completed before said requirement itself can be finished. As this is the case the following represents the order of importance for each functional requirement.

Requirement	**Dependant Requirements**
FR1	FR2, FR3, FR4, FR5, FR6, FR7, FR8, FR9 ' FR10, FR11, FR12, FR13, FR14, FR15, FR16
FR2	FR3, FR6, FR7, FR11, FR12, FR13, FR14, FR15
FR3	FR6, FR7, FR11, FR12, FR13, FR14, FR15.
FR8	FR7, FR11, FR13
FR4	FR5
FR5	FR11
FR10	FR5
FR12	FR11
FR13	FR6
FR7	FR11
FR11	
FR6	
FR9	
FR14	
FR15	
FR16	

Table B.2: Table representing the Functional Requirements and which other requirements are dependent on them.

As described in Table B.2 the number of dependant requirements a requirement has the more important it is to the progress of the application. FR1 is the first task that must be accomplished therefore every other requirement is dependent on this being completed. FR1 is the ability to launch the application, if the application cannot be launched then none of the other requirements can be completed, this is a critical requirement.

FR2 is another critical requirement which must be completed early in development as many other requirements are dependent on its completion. FR2 is the ability to generate a World. A World contains all the data that the algorithm needs in order to both execute and visualise, therefore if there is no way to generate a World there is no way to visualise or model the algorithms execution (FR3).

Apart from FR8 (start the algorithms execution) the remaining functional requirements are somewhat independent of each other and are less critical. However this does not mean that they can be avoided as they will be needed in the final application version.

Annotated Bibliography

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, Nov 1994, pp. 4-6, [Online]. Available: http://www.worldcat.org/isbn/020163361.

> This book is a very good reference point for design patterns in general, providing concise descriptions, examples and use cases. The book examples are implemented using C++, as this will be a Java based project the main concepts are still relevant as the languages aren't too diss-similar.

[2] ——, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, Nov 1994, pp. 127-128, [Online]. Available: http://www.worldcat.org/isbn/020163361.

> This book is a very good reference point for design patterns in general, providing concise descriptions, examples and use cases. The book examples are implemented using C++, as this will be a Java based project the main concepts are still relevant as the languages aren't too diss-similar.

[3] Mono, "Getting Started — Mono , [online] Feb 2015," Avaliable: http://www.mono-project.com/docs/getting-started/, accessed: 13 Feb 2015.

> Mono provide an open source implementation of the Microsoft .NET framework enabling the development of C Sharp across multiple platforms and environments. The Framework is based on ECMA standards for C Sharp so there is some reliability when developing C Sharp in different environments. However, based on research the implementation of certain features, specifically the newer features in the latest releases of the .NET framework do not always behave as expected. This causes development problems which can easily be avoided for this project.

[4] V. E. Sjoerd, ""Dynamic ant colony optimization for the traveling salesman problem," Master's thesis, Leiden University, The Netherlands, Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands, Jul 2012, pp.7 [Online]. Available: http://www.liacs.nl/assets/2012-08SjoerdvanEgmond.pdf.

[5] Thomas Jungblut, "Ant Colony Optimization for TSP Problems , [online] Feb 2015," Available: http://codingwiththomas.blogspot.co.uk/2011/08/ant-colony-optimization-for-tsp.html, Aug 2011, accessed: 14 Feb 2015.

> Simplistic explanations of each function required. Break down of equations is also included in order to make them more digestible.

[6] Wikipedia, "Ant Colony Optimisation New Pheromone Function , [online] Feb 2015," Avaliable: http://upload.wikimedia.org/math/6/d/b/6db065218c956a4a7af6da99aaeca5d1.png, accessed: 14 Feb 2015.

Simple graphic representing the algera representation of the pheromone fucntion required to update the pheromone matrix.

[7] ——, "Ant Colony Optimisation New Probability Function , [online] Feb 2015," Avaliable: http://upload.wikimedia.org/math/6/d/b/6db065218c956a4a7af6da99aaeca5d1.png, accessed: 14 Feb 2015.

Simple graphic representing the algera representation of the pheromone function required to select the Agents next movements.

[8] ——, "Ant Colony Optimisation Pheromone Function , [online] Feb 2015," Avaliable: http://upload.wikimedia.org/math/e/1/3/e1320f5f72b21e5766dfa7e29b536883.png, accessed: 14 Feb 2015.

Simple graphic representing the algebra representation of the pheromone function required to update the pheromone matrix.