Visualising Ant Colonoy Optimisation

Design

Version: 1.0 Draft
G400 Computer Science, CS39440

Author: Christopher Edwards
che16@aber.ac.uk

Supervisor: Dr. Neil Mac Parthalin
ncm@aber.ac.uk

An overview of design and design rationale for a Computer Science Major Project

Department of Computer Science
Aberystwyth University
Wales

1 Introduction

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.

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.

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.

3.1 Design and Architectural Patterns

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.

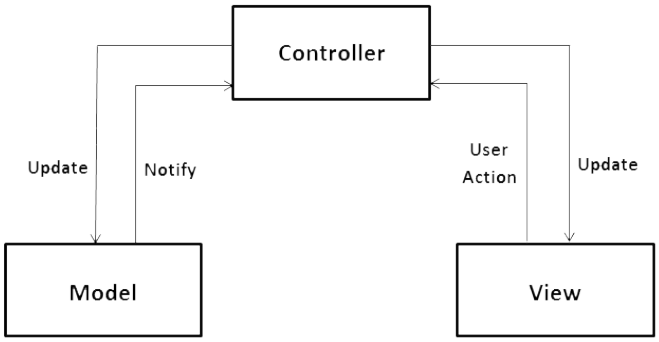

Figure 1: Basic abstract overview of the Model-View-Controller pattern

- **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).

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.

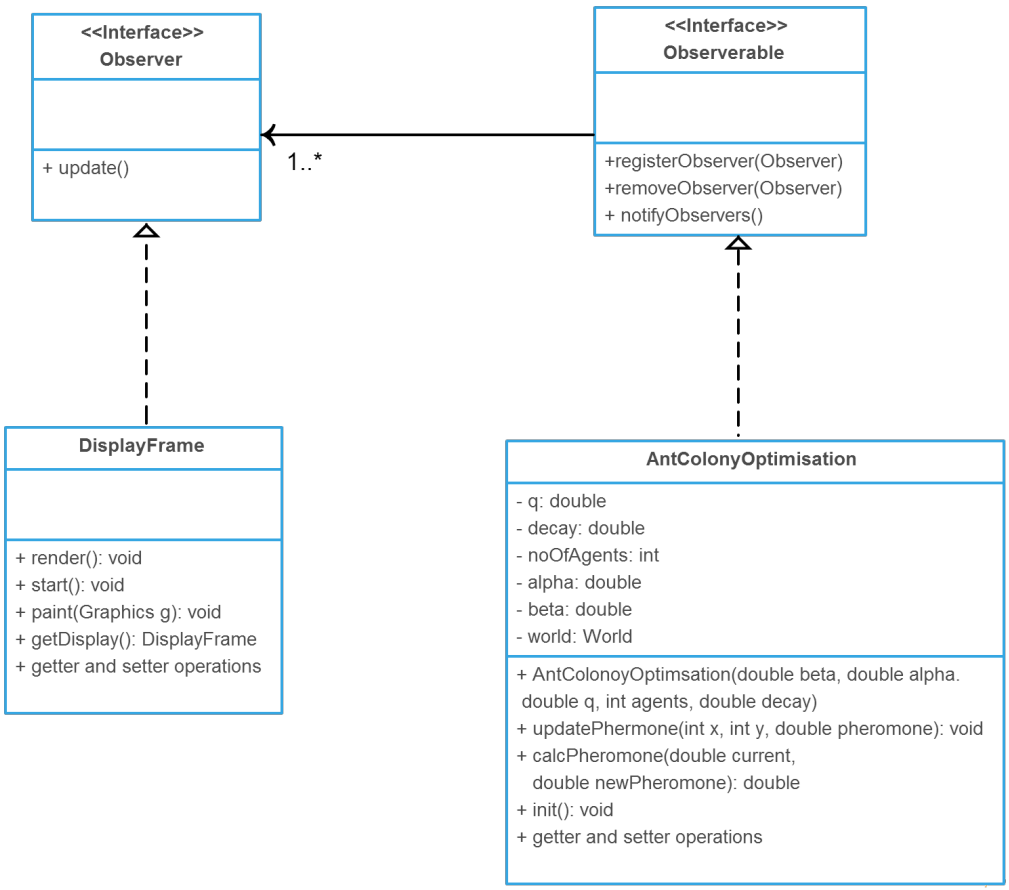

Figure 2: Proposed implementation of the Observer and Observable Design pattern

As figure 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.

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.

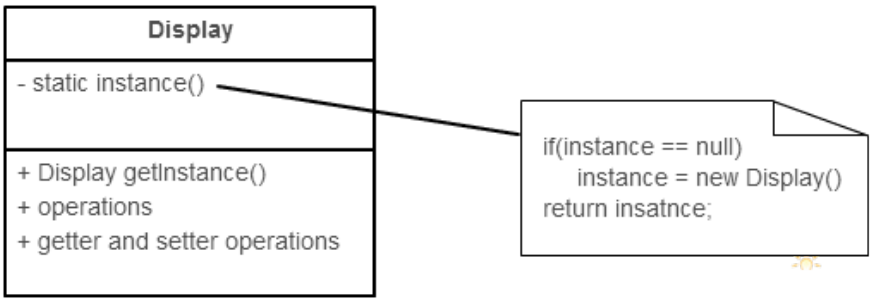

Figure 3: Proposed high-level implementation of the Singleton pattern demonstrating how the sole instance of the graphical user interface will be tracked.

3.3 Structure

Adhering to the concepts of the patterns described in sections 3.1.1 and 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 4 and is represented using standard UML notation.

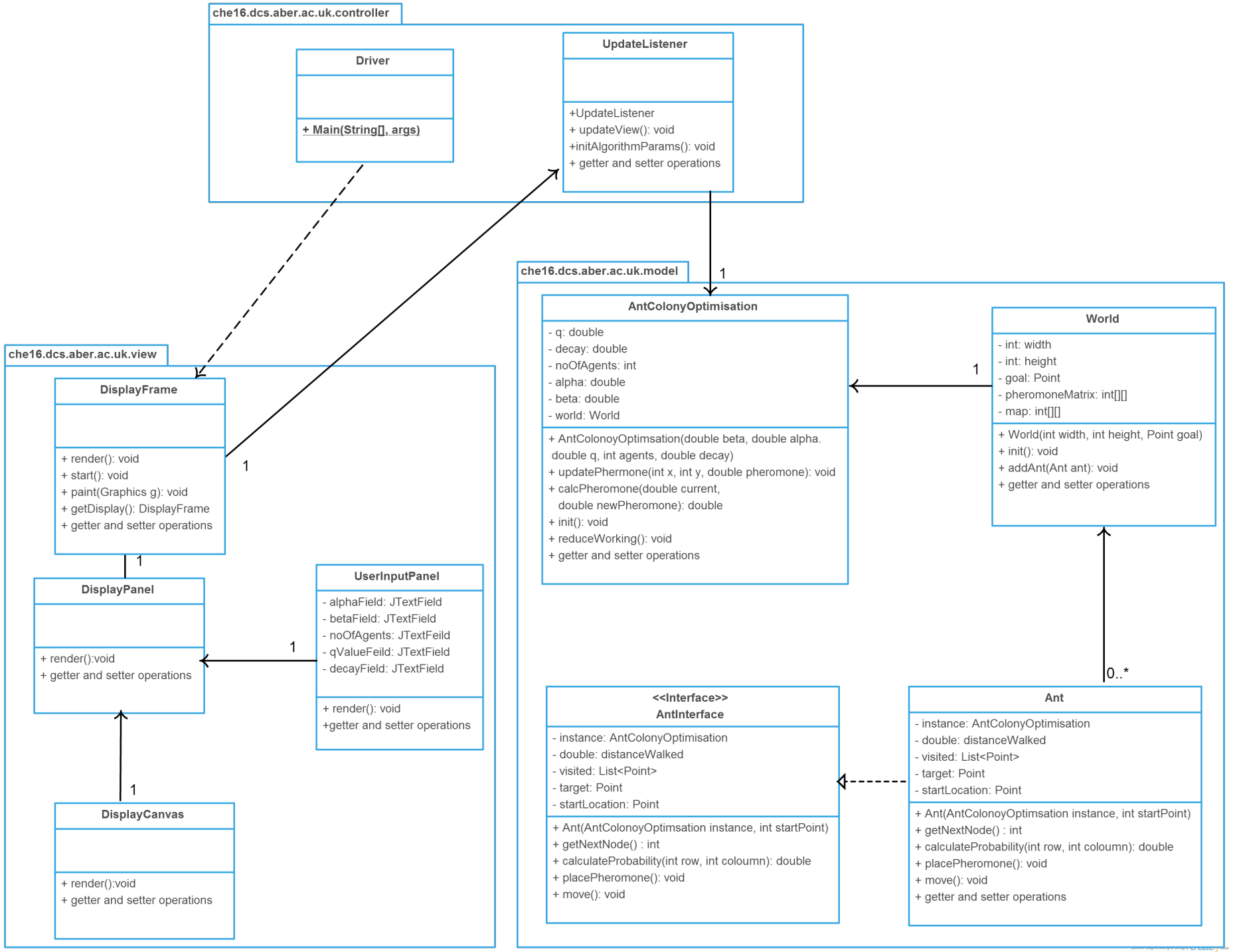

Figure 4: Initial Class Diagram representing the main modules, packages and system interactions using UML notation.

3.3.1 View Package

The *che*16.*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 3.3.2. Instead the *che*16.*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 3.1.1.

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 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 3.1.1.

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.

4 Interface Design

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.

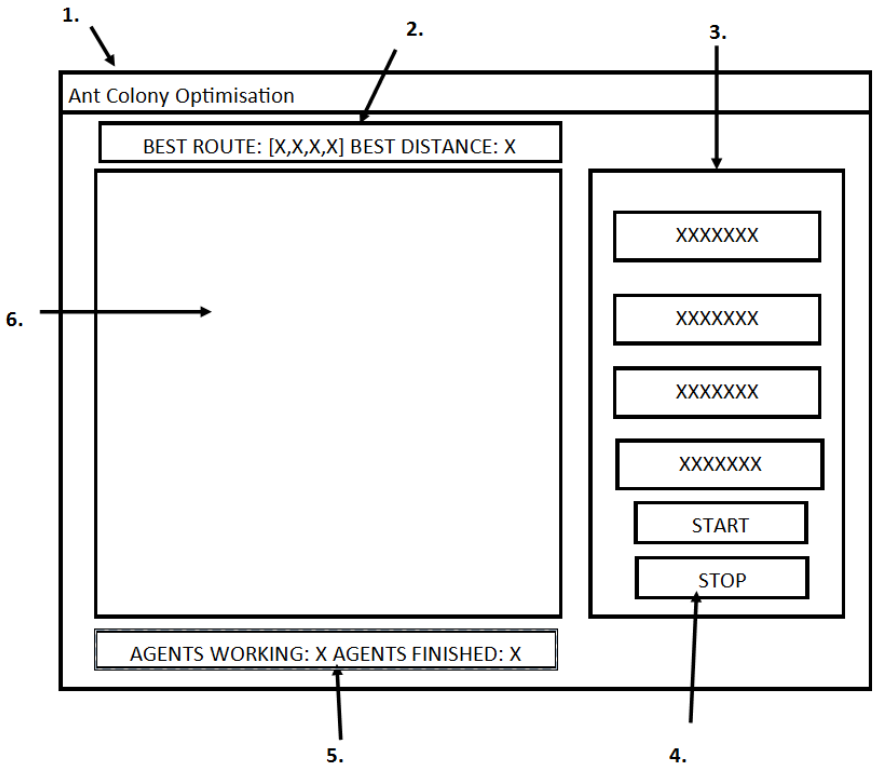

Figure 5: Proposed design for the graphical user interface, showing the key elements in their planned locations.

1. in figure 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 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 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 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 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 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 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.

4.2 Error Message Feedback

As the Algorithm parameters will be user defined using the interface proposed in section 4.1, figure 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.

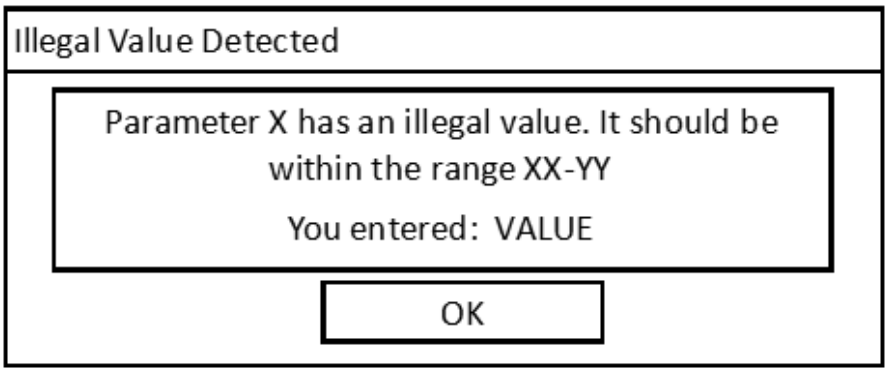

Figure 6: Proposed design for the illegal parameter error message displayed to the user

As represented in figure 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 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.

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 5.2.2. The Agent's next move is influenced by the result of the equation stated in section 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 5.2.1).

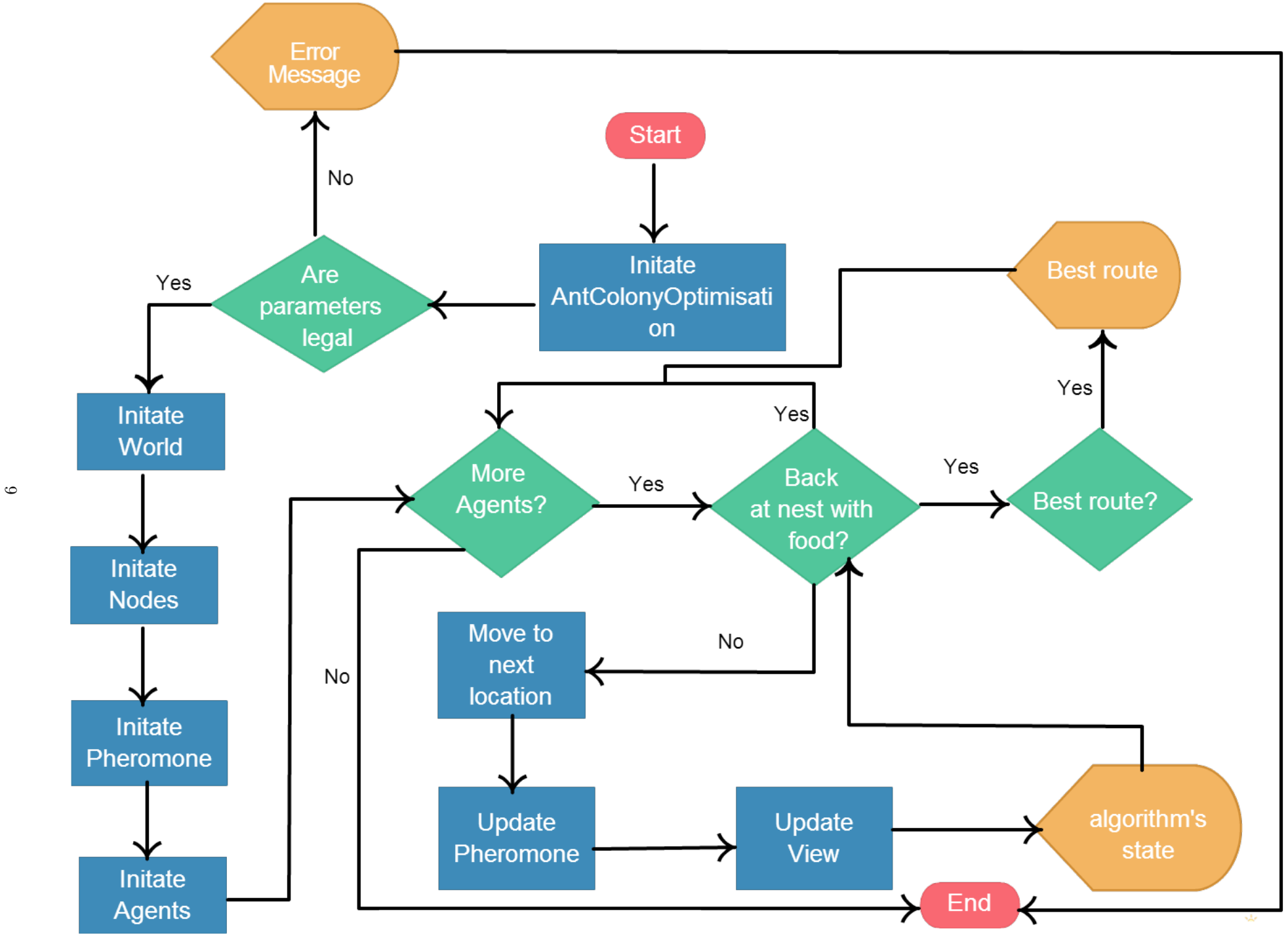

Figure 7: High level flow digram representing the psudeo code for Algorithm 1, section 5.1

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 5.2. The final algorithm may differ from the above depending on any additional feature present in the final release.

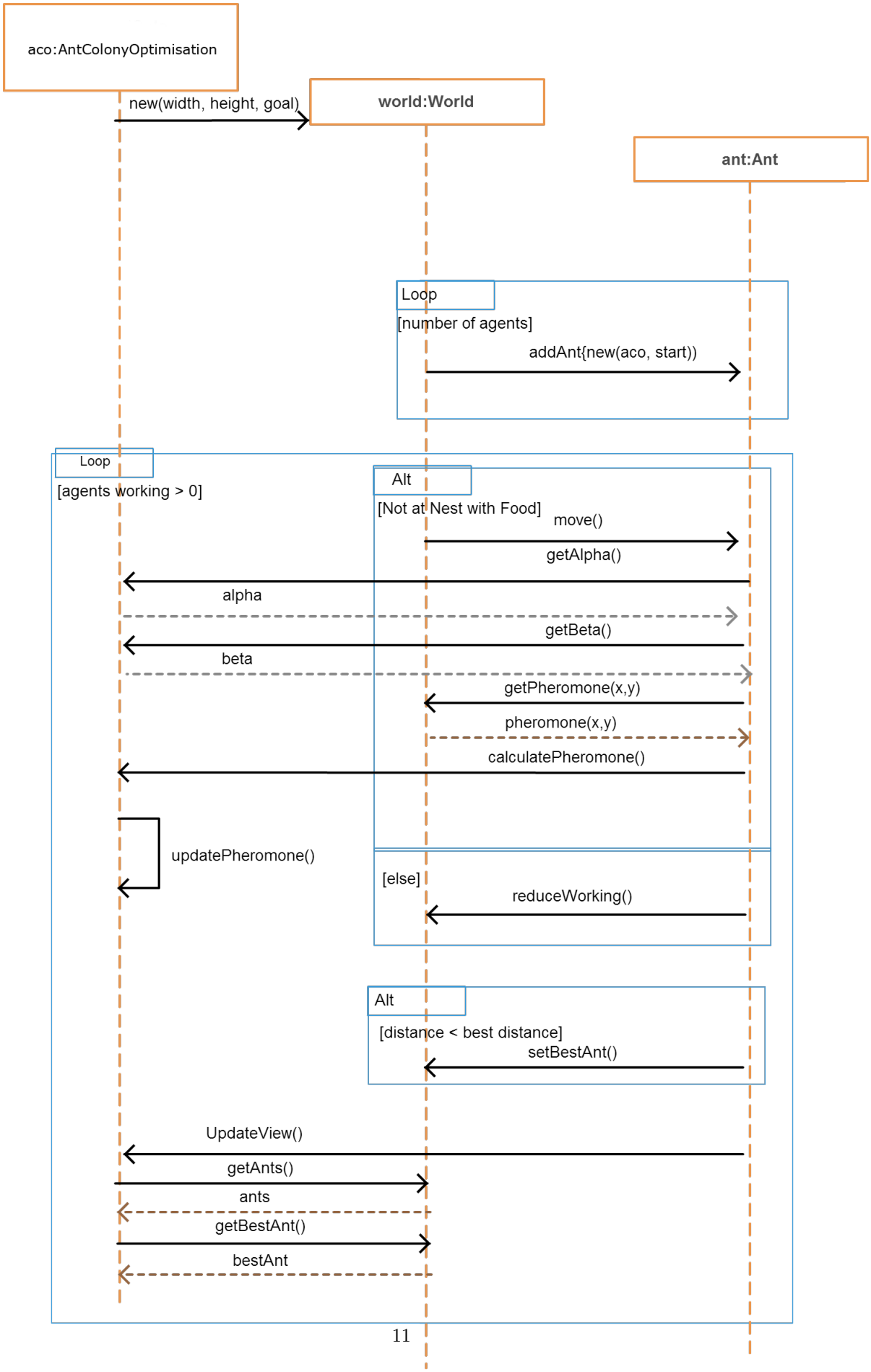

Figure 8: Sequence diagram representing Algorithm 1, section 5.1

The sequence diagram shown in figure 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.

5.2 Metrics

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 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 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: $\quad$ *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 :

5.2.2 Pheromone deposit

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

Figure 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 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 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 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 10, 11

if $pheromoneDeposit = Q\ value/totalDistanceWalked$ ¡ 0 **then**
    $pheromoneDeposit = 0$
$pheromone_{xy} = (1 - algorithm\ decay\ rate) \times currentPheromone_{xy} + pheromoneDeposit$
if $pheromone_{xy} \geq 0$ **then**
    $pheromoneMatrix_{xy} = pheromone_{xy}$
else
    $pheromoneMatrix_{xy} = 0$
end if

6 Representation

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.

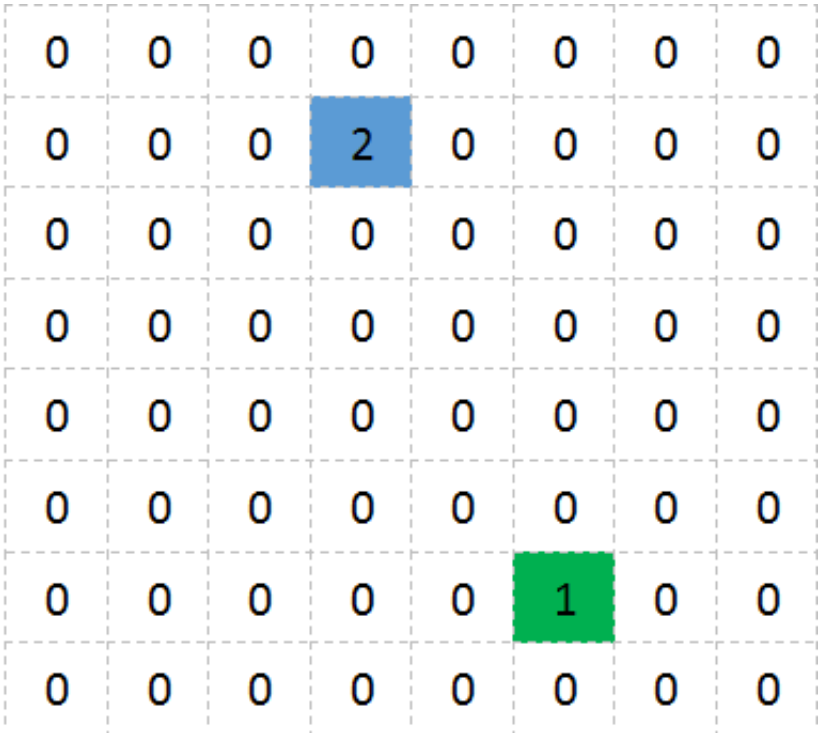

Figure 12: Example World representation using a two-dimensional Integer Array

Figure 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.

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 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.

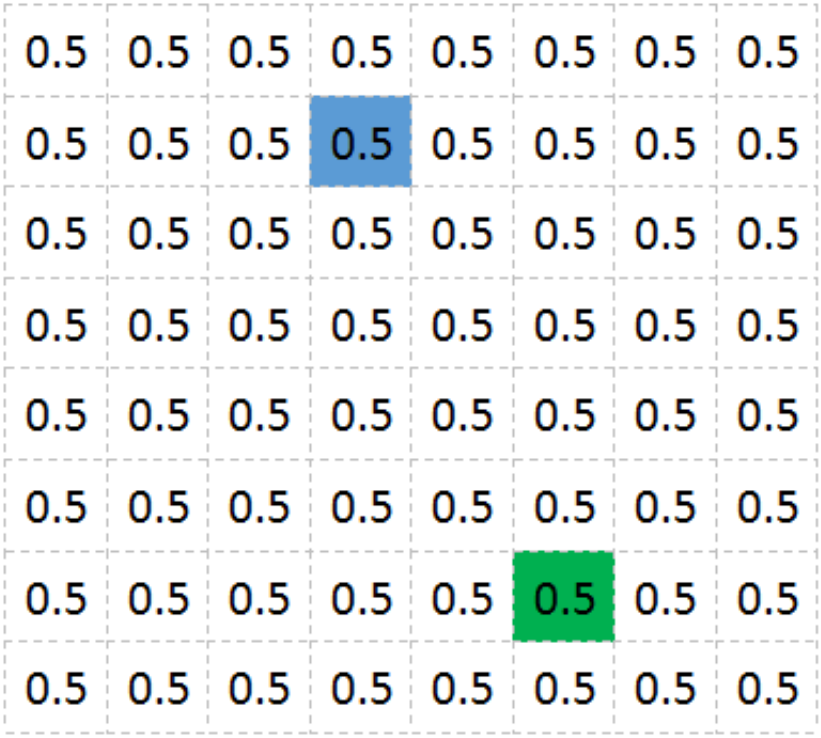

Figure 13: Example Pheromone representation using a two-dimensional Double Array with default initial values

Figure 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 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.

6.3 Visualisation

The data structures described in sections 6.1 and 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 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.

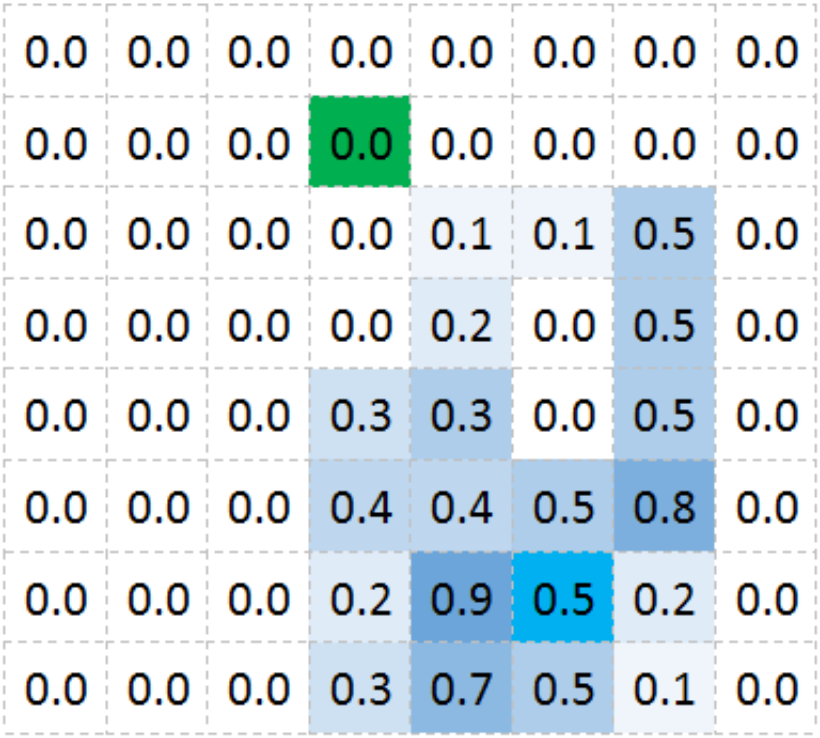

Figure 14: Example Pheromone visualisation

Figure 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.

Bibliography

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 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] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, 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. Avaliable: http://codingwiththomas.blogspot.co.uk/2011/08/ant-colony-optimization-for-tsp.html, Aug 2011. Accessed: 14 Feb 2015.

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

[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] Wikipedia. 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 fucntion required to select the Agents next movements.

[8] Wikipedia. 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 algera representation of the pheromone fucntion required to update the pheromone matrix.