**Requirements Analysis Document**

**Genetic Engine Project**

CITS3200 Professional Computing

2011

University of Western Australia

Crawley, WA, 6009

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Person in charge of Revision | Date of Revision | What was revised? |
| R0.1 | R Gopalan | 08/08/2011 | Created |
| R0.2 | R Gopalan | 16/08/2011 | Modified Section 3.5 |
| R0.3 | B Marshall | 23/08/2011 | Modified Section 3.2 to 3.4 |
| R0.4 | J Hodge | 24/08/2011 | Modified Section 2.0 |
| R0.5 | R Gopalan | 24/08/2011 | Modified Section 1.0 |
| R0.6 | B Marshall | 25/08/2011 | Modified Sections 3.2 to 3.4 |
| R0.7 | B Marshall | 25/08/2011 | Modified Section 3.2 |
| R0.8 | R Gopalan | 26/08/2011 | Modified Section 3.2 |
| R0.9 | R Gopalan | 08/09/2011 | Modified Meeting Dates |

# Client Sign-off

I, Peter Thonell have read the Requirements Analysis Document and have agreed that the information provided by the Genetic Engine Project Team is accurate according to my own needs. By signing this document, I also agree that the prototypes provided by this team, are also accurate as possible to my requirements

Signed:

Date: / /

If there are any issues, with the Requirements Analysis Document and the prototypes, please attach suggestions for improvement to this document.

# Preface:

This document addresses the requirements of the Genetic Engine system. The intended audience for this document are the designers and the client of the project.

# Target Audience:

Client, Developers

# CITS3200 Group J Members:

|  |  |
| --- | --- |
| Group Member | Main Role |
| Rohit Gopalan | Project Leader and User Interface Developer |
| John Hodge | API Developer |
| Brian Marshall | API Developer |
| Alwyn Kyi | API/User Interface Tester |
| Antriksh Srivastava | API/User Interface Tester |

Client:

Mr Peter Thonell

# Meeting Times

* Group Meeting was held on 08/08/2011, 10am at Hacket Hall Café, University of Western Australia
* Client Meeting was held on 08/08/2011, 11am at Hacket Hall Café, University of Western Australia
* Group Meeting was held on 15/08/2011, 1pm at Lab 2.01 in CSSE School, UWA
* Client Meeting was held on 17/08/2011, 2pm at Reid Library, UWA
* Group Meeting was held on 22/08/2011, 11am at Hacket Hall Cafe, UWA
* Client Meeting was held on 24/08/2011, 2pm at Reid Library, UWA
* Group Meeting was held on 29/08/2011, 11am at Hacket Hall Cafe, UWA
* Client Meeting was held on 31/08/2011, 2pm at Reid Library, UWA
* Group Meeting was held on 05/09/2011, 11am at Hacket Hall Cafe, UWA
* Client Meeting was held on 07/09/2011, 2pm at Reid Library, UWA
* Group Meeting was held on 12/09/2011, 11am at Hacket Hall Cafe, UWA
* Client Meeting to be held on 14/09/2011, 2pm at Reid Library, UWA

Future Group meeting times will happen every Monday from September 19 2011 until October 17 2011. Venues to be decided later.

Future Client Meeting times will happen every Wednesday from October 5 2011 until October 19 2011. Venues to be decided later.

# KEY MILESTONES

* 26/08/2011 RAD Part 1 (Deliverable A) Due

**1.0 General Goals**

The aim of the system is to provide a general implementation of a genetic algorithm.

This will be done by providing an API through the .Net library in C#.

**2.0 Current System**

There is no current system. We are building a brand new system from scratch.

**3.1 Overview**  
  
The core of this system will be a .Net DLL with the implementation of the genetic engine. It will accept plug-ins to set the behaviour of various stages of the algorithm. These plug-ins will be contained in .Net DLL files and will implement interfaces defined in a support library.

A windows application will provide a graphical user interface to the genetic engine. It will allow the user to select plug-ins and set various parameters then run the algorithm.

A set of sample plug-ins are also required to solve a problem given by the client: Given a map with the locations of a number of towns and start and end points produce a network of roads which includes the start and end points and balances two goals:

1. Minimise the total length of the roads.
2. Minimise the distance of each town from its closest vertex in the network.

The generations produced when the engine is used with these sample plug-ins will be written to a file. A visualisation tool will be required to load this file and display the individual networks produced on the map.

**3.2 Functional Requirements, their priorities and client Values**

A) Core Genetic Engine Library

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirements** | **Priority** | **Client Value**  **($)** |
| A1 | The system shall provide an engine for running genetic algorithms. | 1 | 10 |
| A2 | The system shall have the ability to load up modules | 3 | 6 |

B) Module Types

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirements** | **Priority** | **Client Value ($)** |
| B1 | The system shall provide a seeding function | 2 | 8 |
| B2 | The system shall provide a genetic operator function which randomly mutates the best road networks in one generation to produce the next. | 1 | 10 |
| B3 | The system shall provide a fitness function which loads a map file and uses it to evaluate the individual road networks. | 2 | 8 |
| B4 | The system shall provide a termination condition function which terminates the algorithm when a certain fitness level is reached. | 4 | 4 |
| B5 | The system shall provide a generator container function | 4 | 4 |

C) Genetic Engine GUI

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirements** | **Priority** | **Client Value ($)** |
| C1 | The system shall give the user the ability to choose any of the module types described above | 4 | 4 |
| C2 | The system should be able to start the GUI | 3 | 6 |
| C3 | The system should be able to stop and continue the generation on the GUI | 5 | 2 |

D) Demo User Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Requirements** | **Priority** | **Client Value ($)** |
| D1 | The system shall provide a sample implementation of the seeding function | 4 | 4 |
| D2 | The system shall provide a sample implementation of the genetic operator function. | 2 | 8 |
| D3 | The system shall provide a sample implementation of the fitness function. | 2 | 8 |
| D4 | The system shall provide a sample implementation of the termination function | 5 | 2 |
| D5 | The system shall be able to save the individual to a specific format which can be viewed by another program. | 3 | 6 |
| D6 | The system shall be able to load the saved individual from a specific directory. | 5 | 2 |
| D7 | The system shall be able to view the individual in a graphical form on the demo application | 2 | 8 |

Note: The estimated time for each requirement is yet to be decided by the team in this stage of the project.

**3.3.1 User Interface and Human Factors**

The GUI should provide a way to quickly select plug-in classes. Once a DLL has been loaded the user should be able to select the plug-in classes easily from a list.

**3.3.2 Documentation**

The users of the genetic engine, sample plug-ins and visualiser tool will be programmers with some experience with C#. Therefore, clear API and source code documentation are the most important source of information. Brief manuals for the GUI and visualiser will also be required

**3.3.3 Hardware Consideration**

The libraries and applications should work on any machine capable of running .Net. Although faster hardware will obviously result in faster solutions.

**3.3.4 Performance Characteristics**

Performance has a lower priority than flexibility and good object oriented code structure however where possible, without sacrificing these, optimisations for speed should be made.

**3.3.5 Error Handling and Extreme Conditions**

The classes in the genetic engine libraries should throw clear and descriptive exceptions when its methods are called incorrectly. These should assist the programmer using these libraries to quickly identify and fix their errors.

The Genetic Engine GUI application should capture the exceptions thrown by the genetic engine library classes and report them in an easy to read format. Where possible, indicating the plug-in this caused the problem.

**3.3.6 System Interfacing**

The chromosome generator and fitness function plug-ins may require input from files and output plug-ins may write to files. However the format of these files will be defined by the implementer of the plug-ins and file I/O will be handled by the plug-ins them.

The sample plug-ins will need to be able to load a map file containing the dimensions of the map, the locations of the towns and the start and end points of the path. The format of this file will be defined by us.

We will also need to define a file format for the sample output plug-in to write. The visualiser will need to be able to load the files produced by this plug-in and extract the individual chromosomes.

**3.3.7 Quality Issues**

The highest priority of the Genetic Engine is its flexibility so it should be able to handle any type of compatible plug-in that is fed into it without errors. Error checking will be implemented to check that the plug-ins is in fact compatible with the engine and written correctly.

There are practically no real visual quality factors to consider and the success of the engine will largely be measured by what goes on 'under the hood', and that it runs through the iterations correctly and outputs the right data.

**3.3.8 System Modifications**

The genetic engine is to be modular. The goal is to have something which can be used in many different ways without having to anticipate these ahead of time.

In defining the plug-in API we should, where possible, avoid assumptions as to the way the user will want to implement these. For example, the writer of an output plug-in may want to display the generation on the screen rather than write it to a file.

Naturally, there will need to be some assumptions made in the definition of the API and implementation of the engine. Therefore, the engine should be written in a way which is easy to extend. Good object oriented programming practices must be followed and the source code will need to be fully documented.

**3.3.9 Physical Environment**

The client plans to run the engine overnight on a standard home PC.

**3.3.10 Security Issues**

Allowing arbitrary libraries to be loaded and the compiled code within them to be executed is a large security risk. It is not an issue if the user is also the one who wrote the libraries. However, if becomes a problem if people are writing and sharing plug-in libraries with people they don't know well. Malicious code could easily be hidden in these libraries and executed without the user's knowledge.

To mitigate this risks the possibility of running the plug-in code with reduced permissions (sandboxing) might be explored.

However, the client has stated that this is not a concern and the use of DLLs should be considered to be at the user's own risk.

**3.3.11 Resource Issues**

After the source code and compiled binaries are supplied to the client he will become responsible for all installation and maintenance.

**3.4 Constraints**

The project is to be developed in C# using Visual Studio or MonoDevelop.

**3.5 System Model**

You will have to use the UML (Unified Modelling Language) to create the models. If the CASE tools is not installed yet (Together-J), you can use Visio or PowerPoint to produce the models. For more information on the notations of UML, check out the following rational websites - [Notation](http://www.rational.com/uml/html/notation/) and [Documentation](http://www.rational.com/uml/documentation.html/). To make your models more readable, you have to include some texts to guide the reader along the flow of your model. These texts are called Navigational Text because they help to move the reader along the models.

**3.5.1 Scenarios**

For this section, think about all the possible ways which the users will interact with your subsystem. Present them in a "story" format.

**3.5.2 Use Case Models**

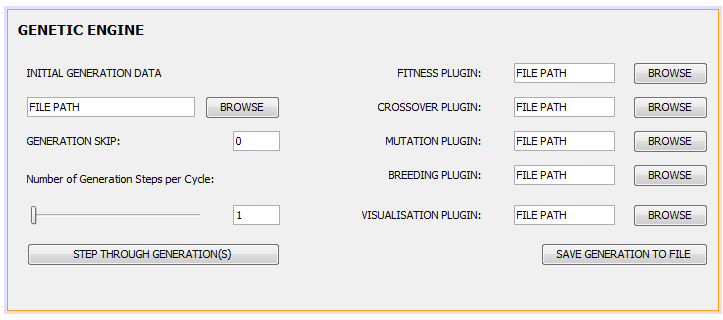
3.5.2.1 Actors   
3.5.2.2 Use Cases

**3.5.3 Object Models**

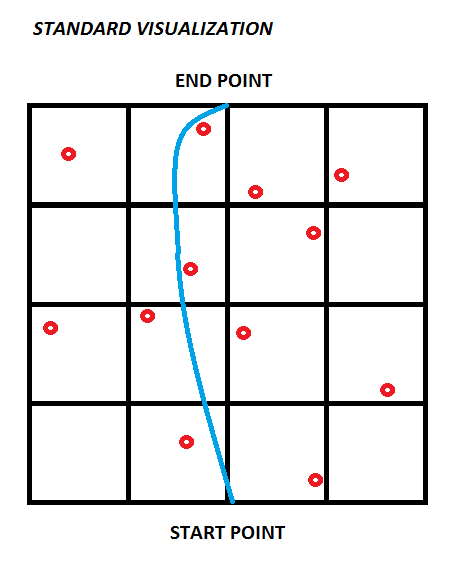
3.5.3.1 Data Dictionary   
3.5.3.2 Class Diagrams

**3.5.4 Dynamic Models**

**3.5.5 User Interface - Navigational Paths and Screen Mock-ups**

****

**Figure 3.1** The screen mock-up of the Genetic Engine Interface. Each plugin is to be specified by their file-path.



**Figure 3.2** Pathfinding using Genetic Algorithms shown visually in Grid view

**4.0 Glossary**