**Requirements Analysis Document**

**Genetic Engine Project**

CITS3200 Professional Computing

2011

University of Western Australia

Crawley, WA, 6009

**Revision History:**

Version R0.1 08/08/2011 R Gopalan. Created

**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:**

Rohit Gopalan (Project Leader), John Hodge, Alwyn Kyi, Brian Marshall, Antriksh Srivastava

**Client:** Mr Peter Thonell

**Meeting Times (Past and upcoming 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

**MILESTONES**

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

**CLIENT SIGN OFF:**

**1.0 General Goals**

For this section, enter the goals of your subsystem, i.e. what are the objectives of the functions of your subsystem?

**2.0 Current System**

For this section, describe the current situation that is relevant to your subsystem.

**3.0 Proposed System**

**3.1 Overview**  
  
The Genetic Engine will provide a general implementation of a genetic algorithm. It will configurable and extensible to run a variety of specific genetic algorithms.

An example implementation of an genetic algorithm is also required. This will accept a map with the locations of towns and the start and end points of a path. It will generate a path which balances two requirements. These requirements are to minimize the total path length and to minimize the distance of each town from its closest point on the path. This will include a tool to visualise individual paths generated by the algorithm.

**3.2 Functional Requirements**

.Net Library (DLL) written in C# containing:

* Core genetic engine class:
  + Maximum flexibility
  + Accept arbitrary class as chromosome.
  + Configurable via plug-ins for fitness function, mutation and conjugation.
  + Write the best individuals of each generation to a file.
* Built-in plug-ins:
  + 2 Mutation Operators
  + 2 Conjugation Operators

Example tool written in C#:

* Application using genetic engine DLL:
  + Read map file:
    - Width as an integer
    - Height as an integer
    - List of town coordinates as pairs of integers.
    - Path start and end coordinates as pairs of integers.
  + Initialise Genetic Engine with suitable plug-ins and run algorithm
  + Output best path.
* Visualisation tool:
  + Read individual path from file.
  + Display path on a map with the town locations.

Note:

Paths will be in the form of trees. That is, each path will be an undirected graph which is connected and has no cycles. For simplicity only the vertices in the tree (not the edges) will be used when determining the minimum distance of the path from each town.

An undirected graph would have been a more general representation of the path however, any graph can be reduced to a tree by removing edges. The resulting tree will contain all the same vertices, remain connected and have total length less than or equal to the original graph. This simplifies the algorithm as it is much easier to define conjugation operations for trees than for graphs.

**3.3.1 User Interface and Human Factors**

The users of the genetic engine library, sample application and visualiser tool will be programmer with some experience with C#. Therefore, clear API and source code documentation are the most important source of information.

The sample application has little practical value in itself other than proof that the genetic engine library works. However, it's source code will serve as an example of how to utilise the classes within the library.

**3.3.2 Documentation**

Genetic Engine library:

API documentation outlining all exposed classes and how they are intended to be used.

Tutorial document with step-by-step instructions for a simple example application using the library.

Clear and complete source code documentation.

Sample Application and Visualisation Tool:

Usage instructions

Clear and complete source code documentation.

The users of the sample application and visualiser tool are also likely to be programmers. The sample application will be a command-line utility and therefore require clear documentation of the command-line options.

**3.3.3 Hardware Consideration**

The DLL should work on any PC with support for .Net. The sample application and visualiser tool should run on any Windows PC with support for .Net.

**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 genetic engine should throw clear and descriptive exceptions when its methods are called incorrectly.

**3.3.6 System Interfacing**

For this section, think about the I/O of your subsystem. Consider the following:   
Is input coming from systems outside the proposed system? Is output going to systems outside the proposed system? Are there restrictions on the format or medium that must be used for input or output?

**3.3.7 Quality Issues**

For this section, focus on the possible quality enhancement or compromises. Consider the following:   
What are the requirements for reliability? Must the system trap faults? Is there a maximum acceptable time for restarting the system after a failure? What is the acceptable system downtime per 24-hour period? Is it important that the system be portable (able to move to different hardware or operating system environments)?

**3.3.8 System Modifications**

For this section, think about the current infrastructure of your system which will be extended for future features, incorporated or made obsolete. Consider the following:   
What parts of the system are likely candidates for later modification? What sorts of modifications are expected?

**3.3.9 Physical Environment**

For this section, consider the physical environment in which your subsystem will exist. Consider the following:   
Where will the target equipment operate? Will the target equipment be in one or several locations? Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibrations, magnetic fields ...)?

**3.3.10 Security Issues**

For this section, focus on all possible security considerations. Consider the following:   
Must access to any data or the system itself be controlled? Is physical security an issue?

**3.3.11 Resource Issues**

For this section, think about data management for your subsystem. Consider the following:   
How often will the system be backed up? Who will be responsible for the back up? Who is responsible for system installation? Who will be responsible for system maintenance?   
.

**3.4 Constraints**

For this section, consider all the limitations imposed on your subsystem. Consider the following:   
Constraints on the programming language. Constraints on the development environment. Constraints on the use of libraries. Constraints on the use of legacy systems.

**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 text 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**

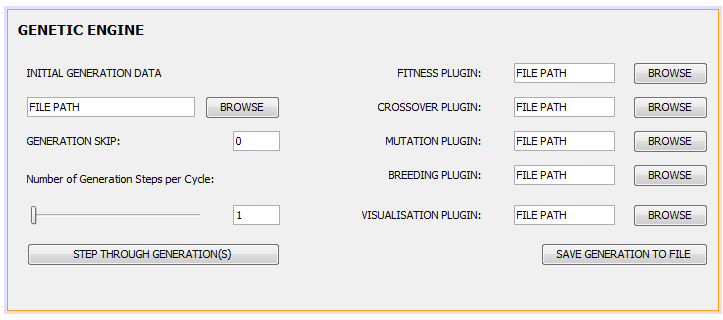
3.5.2.1 Actors   
3.5.2.2 Use Cases

**3.5.3 Object Models**

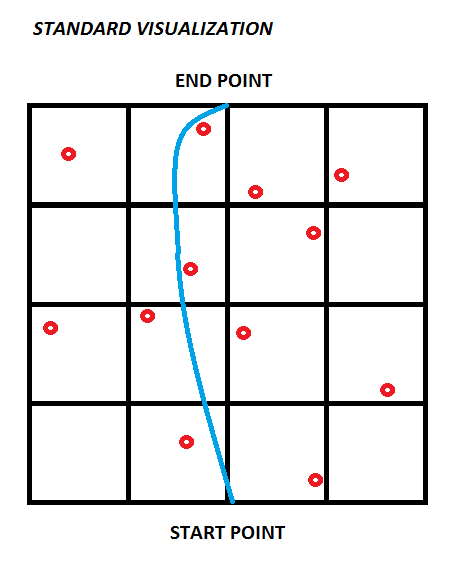
3.5.3.1 Data Dictionary   
3.5.3.2 Class Diagrams

**3.5.4 Dynamic Models**

**3.5.5 User Interface - Navigational Paths and Screen Mockups**

****

**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 Gridview

**4.0 Glossary**