**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
* Group Meeting was held on 15/08/2011, 1pm at Lab 2.01 in CSSE School, UWA
* Client Meetng was held on 17/08/2011, 2pm at Reid Library, UWA
* Group Meeting to be held on 22/08/2011, 11am at Hacket Hall Cafe, UWA
* Client Meeting to be held on 24/08/2011, 2pm at Reid Library, UWA

**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 a 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**

Core .Net Library (DLL) written in C# exposing:

* Genetic engine class:
  + Maximum flexibility
  + Accept arbitrary class as chromosome.
  + Configurable via plug-ins for initial population, fitness function, genetic operators and termination condition.
  + Write the best individuals of each generation to a file.
* Plug-in Loader
  + Load plug-in classes from user-specified .Net DLL files.
  + Create instances of plug-in classes for use in the genetic engine.

Support .Net Library (DLL) written in C# exposing:

* Interfaces to be implemented by plug-in classes
* Other classes required by plug-in classes

Command-line application written in C#:

* Reads XML configuration file identifying the plug-ins and other parameters to run the algorithm with.
* Loads plug-ins from .Net DLL files.
* Runs the genetic engine with the selected plug-ins and parameters.

Sample plug-ins and configuration files for the path optimisation problem:

* Chromosome Generator: loads map file and generates random paths in the map:
* Fitness function: Assigns higher fitness to paths which are shorter and approach closer to the cities.
* Genetic operators
  + 2 Path mutation operators
  + 2 Path conjugation operators
* Terminator

Visualisation Application:

Load paths from text file outputted by genetic engine when run with example plug-ins.

Display the path on its map.

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, 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, 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.

The sample plug-ins have little practical value in themself other than proof that the genetic engine library works. However, their source code will serve as an example of how to utilise the classes within the library.

**3.3.2 Documentation**

Core genetic engine library and support 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.

Command-line application:

* Explanation of all command line options with examples
* Explanation of XML configuration file format with examples

Sample Plug-ins and Visualisation Tool:

* Usage instructions
* Clear and complete source code documentation.

**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 command-line application should identify problems with the configuration as early as possible and report it in a clear format, identifying the items which caused the problem and explaining why they are invalid. It should also capture the exceptions thrown by the genetic engine library classes and report them in an easy to read format, indicating the plug-in which caused the problem.

**3.3.6 System Interfacing**

The genetic visualiser tool must be able to read the format produced by the genetic engine when used with the sample plug-ins.

User-defined plug-in libraries must conform to the interfaces defined in the support library.

This software will not interface with any other systems (other than the operating system for file access, DLL loading and console I/O)

**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**

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) should be explored.

**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**

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 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**

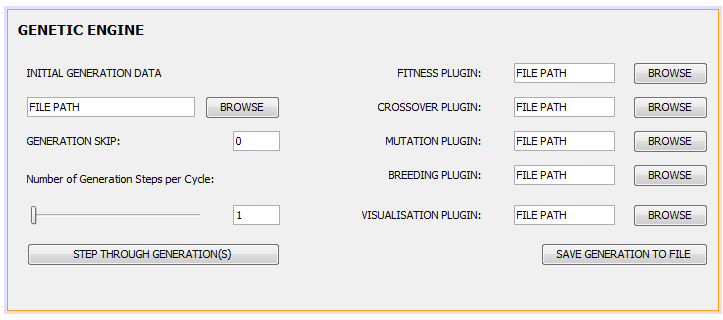
3.5.2.1 Actors   
3.5.2.2 Use Cases

**3.5.3 Object Models**

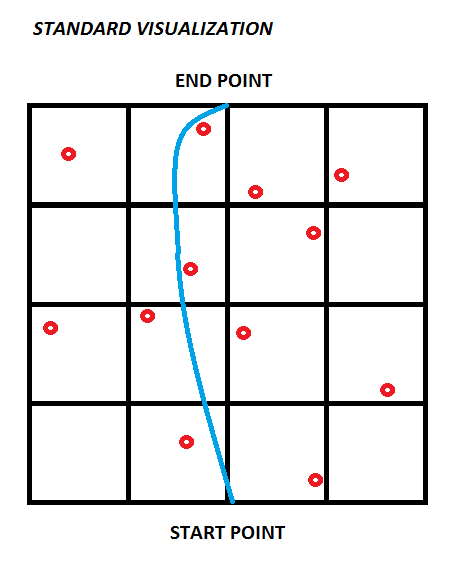
3.5.3.1 Data Dictionary   
3.5.3.2 Class Diagrams

**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**