

# GENETIC ENGINE PROJECT


## REQUIREMENTS ANALYSIS DOCUMENT

---

*Rohit Gopalan, John Hodge, Alwyn Kyi, Brian Marshall, Antriksh Srivastava*

Client: *Mr Peter Thönnell*

CITS3200 Professional Computing 2011  
University of Western Australia Crawley, WA, 6009



```
def getSolutionCosts (navigationCode):  
    fuelStopCost = 15  
    extraComputationCost = 8  
    thisAlgorithmBecomingSkynetCost = 999999999  
    waterCrossingCost = 45
```

GENETIC ALGORITHMS TIP:  
*ALWAYS* INCLUDE THIS IN YOUR FITNESS FUNCTION

## Revision History

Version	Author	Date	Reason
r1	R. Goplan	08/08/2011	Created
r2	R. Goplan	16/08/2011	Modified 3.5
r3	B. Marshall	32/08/2011	Modified 3.2 - 3.4
r4	J Hodge	24/08/2011	Modified Section 2.0
r5	R Gopalan	24/08/2011	Modified Section 1.0
r6	B Marshall	25/08/2011	Modified Sections 3.2 to 3.4
r7	B Marshall	25/08/2011	Modified Section 3.2
r8	R Gopalan	26/08/2011	Modified Section 3.2
r9	R Gopalan	08/09/2011	Modified Meeting Dates
r10	J Hodge	12/09/2011	LaTeX-ify
r11	R Gopalan	13/09/2011	Modified Section 3.2

## 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

## 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 Café, 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 Café, 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 Café, 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 Café, 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.

## Contents

<b>1</b>	<b>General Goals</b>	<b>4</b>
<b>2</b>	<b>Current System</b>	<b>4</b>
<b>3</b>	<b>Proposed System</b>	<b>4</b>
3.1	Overview . . . . .	4
3.2	Functional Requirements, their priorities and client values . . . . .	4
3.3	Non-Functional Requirements . . . . .	5
3.3.1	User Interface and Human Factors . . . . .	5
3.3.2	Documentation . . . . .	5
3.3.3	Hardware Consideration . . . . .	5
3.3.4	Performance Characteristics . . . . .	5
3.3.5	Error Handling and Extreme Conditions . . . . .	5
3.3.6	System Interfacing . . . . .	5
3.3.7	Quality Issues . . . . .	6
3.3.8	System Modifications . . . . .	6
3.3.9	Physical Environment . . . . .	6
3.3.10	Security Issues . . . . .	6
3.3.11	Resource Issues . . . . .	6
3.4	Constraints . . . . .	6
3.5	System Model . . . . .	6
3.5.1	Scenarios . . . . .	6
3.5.2	Use Case Models . . . . .	7
3.5.3	Object Models . . . . .	7
3.5.4	Dynamic Models . . . . .	12
3.5.5	User Interface - Navigational Paths and Screen Mockups . . . . .	13
<b>4</b>	<b>Glossary</b>	<b>14</b>

# 1 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 Current System

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

# 3 Proposed System

## 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

### 1. Core Genetic Engine Library

No.	Requirements	Priority	Client Value (\$)	Hours (est.)
1.1	The system shall provide an engine for running genetic algorithms.	1	10	12
1.2	The system shall have the ability to load up modules	3	6	12

### 2. Module Types

No.	Requirements	Priority	Client Value (\$)	Hours (est.)
2.1	The system shall provide a seeding function	2	8	6
2.2	The system shall provide a genetic operator function which randomly mutates the best road networks in one generation to produce the next.	1	10	18

### 3. Genetic Engine GUI

No.	Requirements	Priority	Client Value (\$)	Hours (est.)
3.1	The system shall give the user the ability to choose any of the module types described above	4	4	10
3.2	The system should be able to start the GUI	3	6	10
3.3	The system should be able to stop and continue the generation on the GUI	5	2	10

## 4. Demo User Interface

No.	Requirements	Priority	Client Value (\$)	Hours (est.)
4.1	The system shall provide a sample implementation of the seeding function	4	4	4
4.2	The system shall provide a sample implementation of the genetic operator function.	2	8	4
4.3	The system shall provide a sample implementation of the fitness function.	2	8	4
4.4	The system shall provide a sample implementation of the termination function	5	2	4
4.5	The system shall be able to save the individual to a specific format which can be viewed by another program.	3	6	16
4.6	The system shall be able to load the saved individual from a specific directory.	5	2	16
4.7	The system shall be able to view the individual in a graphical form on the demo application	2	8	20

### 3.3 Non-Functional Requirements

#### 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 which caused the problem.

#### 3.3.6 System Interfacing

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.

### 3.3.7 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, it 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

### 3.5.1 Scenarios

Since this is an API, there is not much for the scenarios

#### 3.5.1.1 Scenario 1

A client wants to solve a sudoku using a simple genetic engine (random initial state, random breeding). They implement the generator to create a random set of input values, a fitness function that checks how close to a valid solution the current item is, a breed function to then perform weighted breeding on the population (the higher a solution's fitness value, the more likely it is to breed) and a termination function (to check if the fitness is 100%). These are then plugged into the API, and the simulation is started.

### 3.5.2 Use Case Models

1. (optional) Load “helper” objects (implementing `IPopulator`, `IEvaluator`, `IGeneticOperator` and `ITerminator`) from a DLL using plugin loader
2. Create instances of each interface
3. Pass created objects to `GeneticEngine`’s constructor
4. Call either `GeneticEngine.Step` (do one generation), `GeneticEngine.Repeat` (do `n` calls to `Step`) or `GeneticEngine.Run` (call `Step` until the terminate says to stop)
5. Get the final generation state using the `GeneticEngine.Generation` accessor, and getting the first (`#0`) element from that array (which is defined to be the individual with the highest fitness).

### 3.5.3 Object Models

#### 3.5.3.1 Data Dictionary

- Namespace: `GeneticEngineCore` (See figures 2 and 6)
  - `GeneticEngine`: Accepts one of each plug-in type and uses them to run the genetic algorithm.
  - `PluginLoader`: Loads a .Net DLL file and creates instances of the classes it contains. This is used to load plug-in classes.
  - `DefaultGenerationFactory`: The implementation of the `IGenerationFactory` interface used by the engine if another is not supplied. This supplies instances of `AATreeGeneration`.
  - `MaxFitnessTerminator`: An implementation of `ITerminator` which terminates the algorithm when an individual is found with fitness greater than or equal to the specified threshold.
- Namespace: `GeneticEnginePlugin` (See figures 3 and 6)
  - `IPopulator`: An implementation of this interface will populate an `ArrayList` representing the initial population of individuals.
  - `IEvaluator`: An implementation of this interface will evaluate an individual, giving its fitness as an unsigned integer.
  - `IGeneticOperator`: An implementation of this interface will process the current generation to produce the next population of individuals.
  - `ITerminator`: An implementation of this interface will process the current generation to determine whether the algorithm is complete.
  - `IOutputter`: An implementation of this interface will output all or part of the current generation. For example it may display the best individual on the screen or write the entire generation to a file.
  - `IGeneration`: An implementation of this interface will hold a set of individuals with their fitness values.
  - `IGenerationFactory`: An implementation of this interface will create empty instances of some implementation of `IGeneration`.
  - `IndividualWithFitness`: Holds an individual with its fitness value. This is used as a return type for lookups in an `IGeneration` so that the individual and its fitness can be returned by a single function call.
- Namespace: `GeneticEngineSupport` (See figures 4 and 6)
  - `AATreeGeneration`: An implementation of the `IGeneration` interface using a self balancing binary search tree. This allows insertion, lookup and deletion of individuals in  $O(\lg(n))$  time. Individuals are stored sorted from highest fitness to lowest and can be accessed by index or by the sum of all fitnesses before the individual.



This second method is intended for use in Fitness Proportionate Selection or Stochastic Universal Sampling. In these, the individuals are selected randomly with probability proportional to fitness. Figure 1 shows 3 individuals: A with fitness 5, B with fitness 3, C with fitness 2. The total fitness is 10. A random number between 0 (inclusive) and 10 (exclusive) will identify an individual: (0-4 gives A, 5-7 gives B and 8-9 gives C).

Figure 1: Fitness Proportionate Selection

A					B			C	
Fitness = 5					Fitness = 3			Fitness = 2	
0	1	2	3	4	5	6	7	8	9

- Namespace: RoadNetworkFinder (See figures 5 and 7)
  - RoadNetworkPopulator: An implementation of the IPopulator interface. This will randomly generate an initial population of RoadNetwork instances.
  - RoadNetworkEvaluator: An implementation of the IEvaluator interface. This will calculate a fitness value for a RoadNetwork based on the total length of road and the distances from each town.
  - RoadNetworkMutationOperator: An implementation of the IGeneticOperator interface. This will generate new individuals by randomly changing the best RoadNetworks in the current generation.
  - RoadNetworkConjugationOperator: An implementation of the IGeneticOperator interface. This will choose pairs from the best individuals and randomly combine their parts to form new individuals.
  - RoadNetworkOutputter: An implementation of the IOutputter interface. This will write the entire generation to a file.
  - RoadNetwork: Represents a network of roads on a given map. This is used as the chromosome type for the RoadNetworkFinder plugins.
  - Map: Represents the map supplied to the algorithm. Defines the extents of the map, start/end points of the road and locations of the towns.
  - Vertex: Represents a vertex in a RoadNetwork.
  - Edge: Represents an edge connecting two vertices in a RoadNetwork.
  - Coordinates: A pair of integers giving an x-y location within a Map.
- Namespace: RoadNetworkFinderGUI (See figure 8)
  - RoadNetworkFinderGUIMainWindow: Provides a graphical front-end to the GeneticEngine and PluginLoader classes for using the RoadNetwork plug-ins.
- Namespace: RoadNetworkVisualiser (See figure 9)
  - RoadNetworkVisualiserMainWindows: A GUI tool to load and view individuals from the files written by RoadNetworkOutputter.

### 3.5.3.2 Class Diagrams

Figure 2: Namespace: GeneticEngineCore

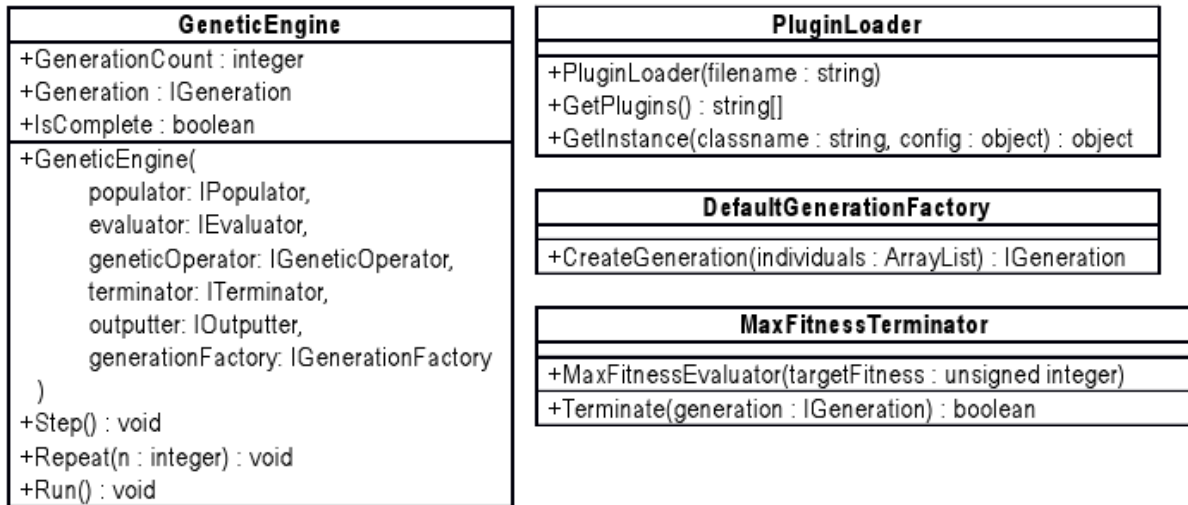


Figure 3: Namespace: GeneticEnginePlugin

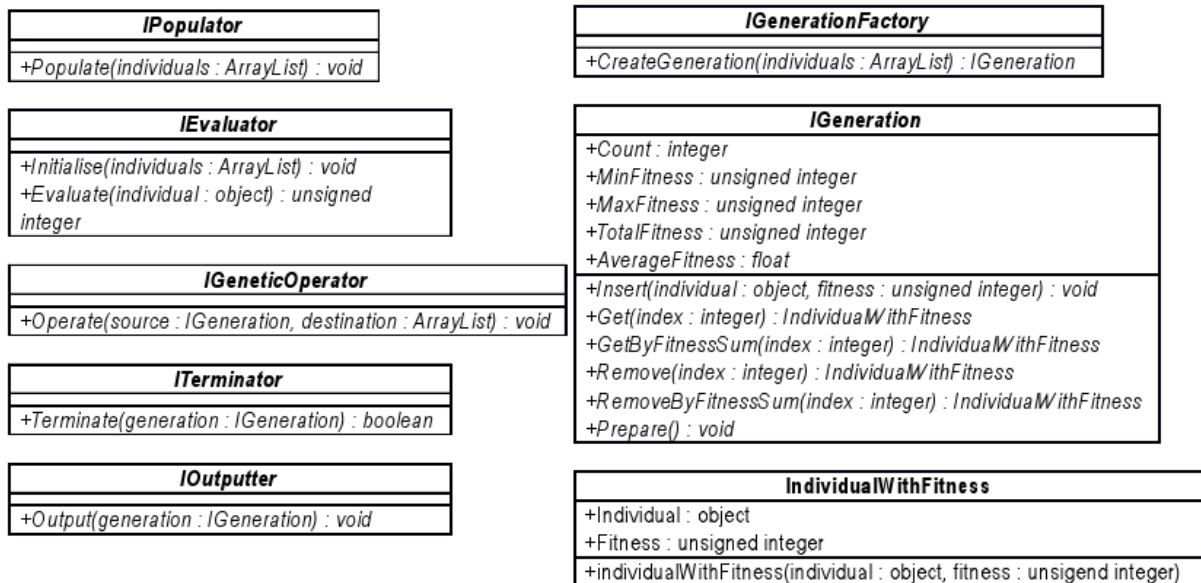


Figure 4: Namespace: GeneticEngineSupport

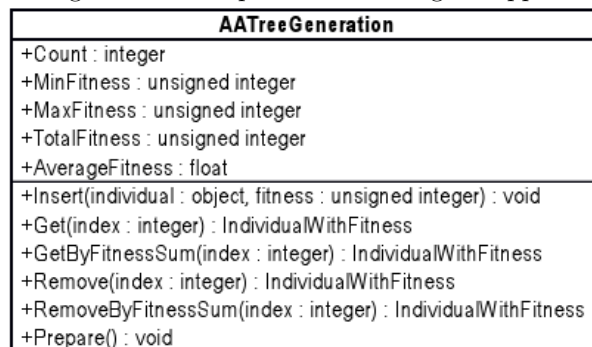




Figure 7: RoadNetworkFinder Dependencies

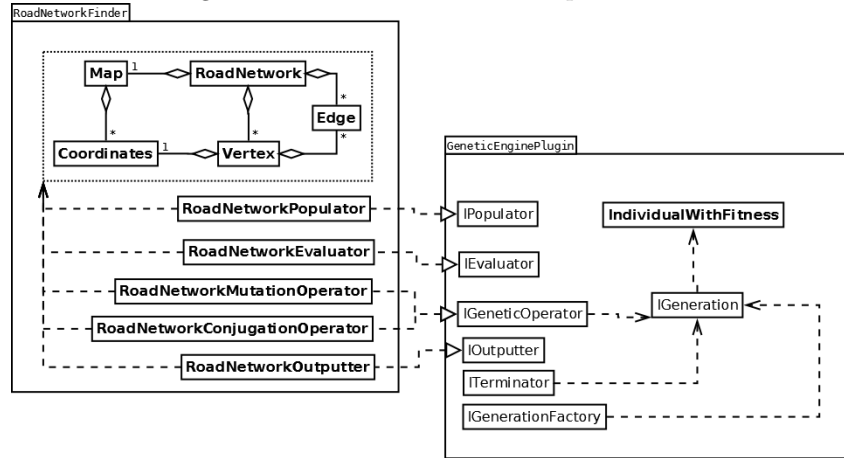


Figure 8: RoadNetworkFinderGUI Dependencies

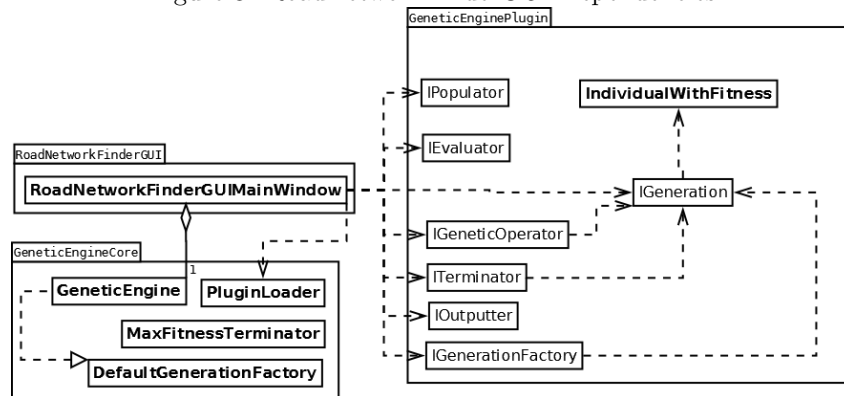
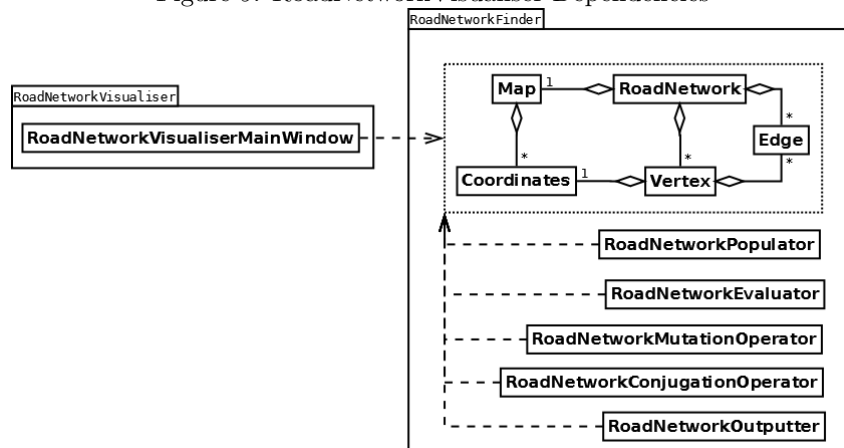
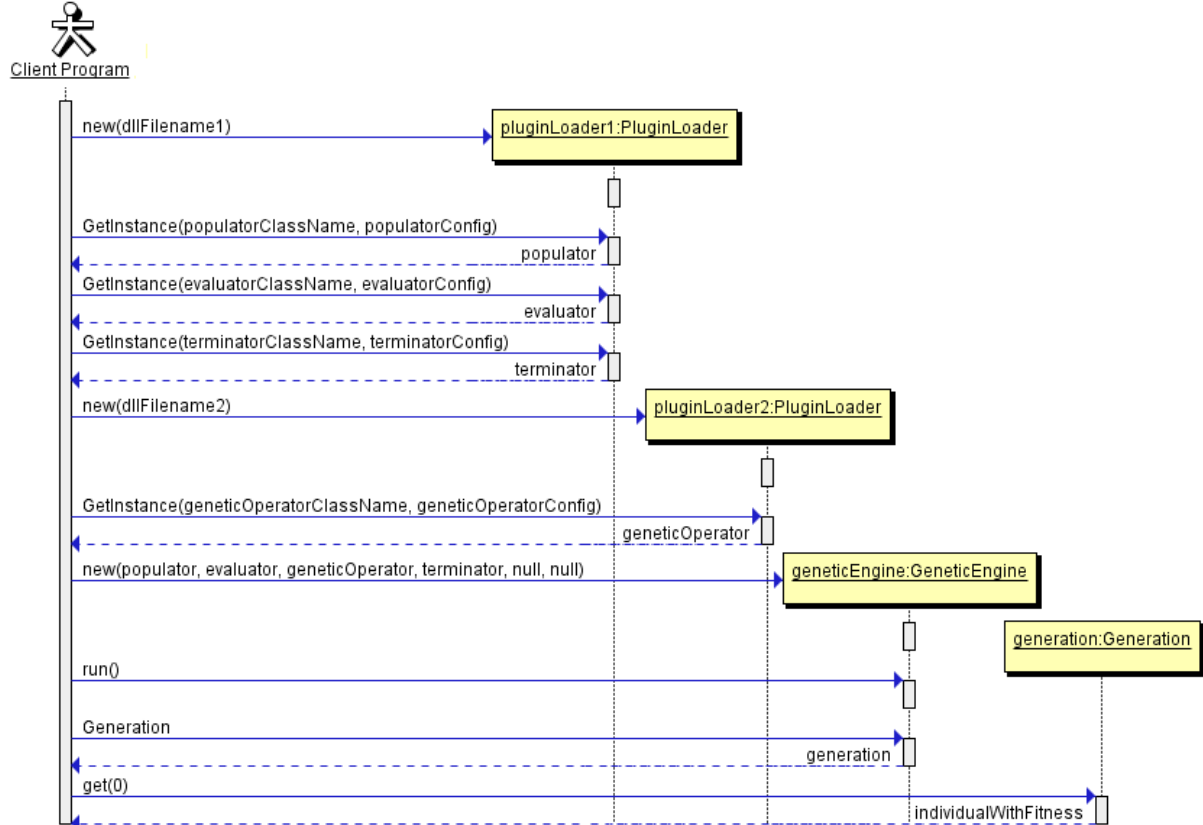


Figure 9: RoadNetworkVisualiser Dependencies



### 3.5.4 Dynamic Models

Figure 10: Sequence Diagram: Initialising and running the GeneticEngine from a client program.



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

Figure 11: Road Network Finder User Interface.

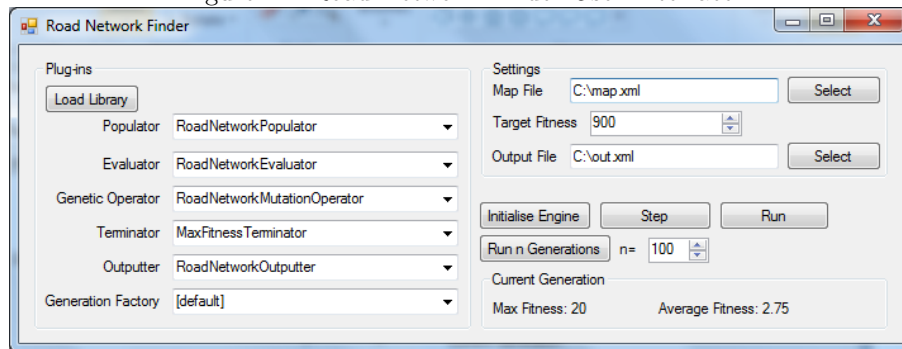
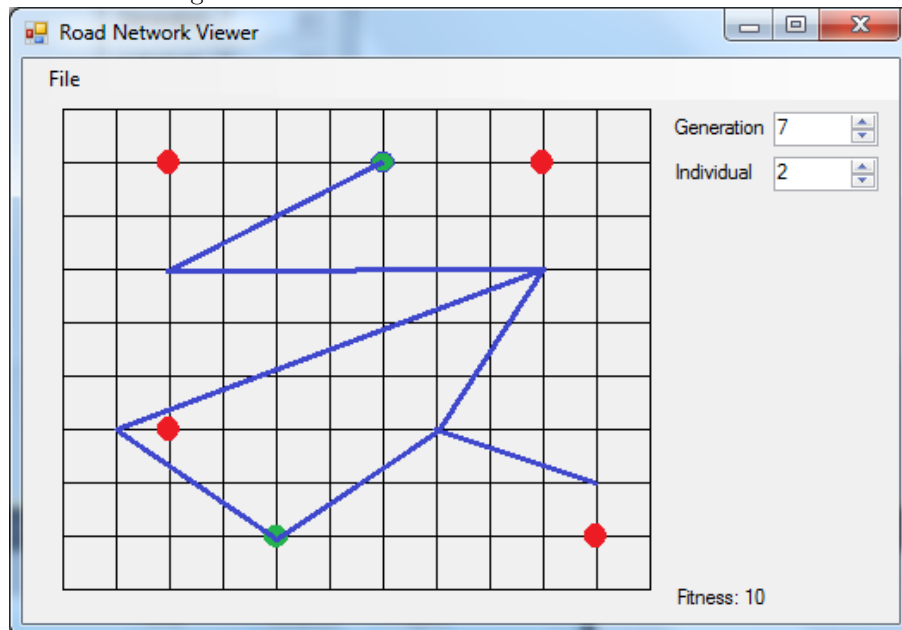


Figure 12: Road Network Visualiser User Interface



## 4 Glossary