Genetic Engine Project
Requirements Analysis Document

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

Client Mr Peter Thörnell

Contents

Revision History

- Version R0.1 08/08/2011 R Gopalan. Created
- Version R0.2 24/Aug/2011 J Hodge. Converted to tex

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

Meeting Times (Past and upcoming times)

- Group Meeting was held on 08/08/2011, 10am at Hacket Hall Caf, University of Western Australia
- Client Meeting was held on 08/08/2011, 11am at Hacket Hall Caf, 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

I, Peter Thonelll have read the documents prepared by the Genetic Engine Project Team and agree that the contents do match the criteria for acceptance
Signed:
Date:

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. This software is not replacing a current implementation, or being used to suppliment known techniques (at least in the client's case).

3 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.

3.2.0.1 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

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 $C\sharp$. 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

3.3.2.1 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.

3.3.2.2 Command-line application

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

3.3.2.3 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 only I/O actually required is the parameters (through the use of plugin files) that are fed into the engine in the beginning. These parameters can be set through an API but may also implement a command line interface and a GUI. After those parameters are set, the program will run its course through a number of iterations.

The output will be set by a visualization plugin if it is implemented in the future, otherwise the data will be saved to a text file.

3.3.7 Quality Issues

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

3.3.8 System Modifications

There is a set specification as to what the underlying engine should do, only minor modifications can be made and only at a higher level (client interface). This includes things such as how the engine accepts input, rather than how it processes it.

3.3.9 Physical Environment

The Genetic Engine will run on a regular computer so no extreme physical conditions will be present.

3.3.10 Security Issues

There are no real security issues with the system however the engine will be encapsulated to ensure that the user does not modify the internal program data and risk a malfunction.

3.3.11 Resource Issues

The engine deals with a set of plugins upon which it will iterate through generations, and then output this to a file. The user is responsible for this file once it has been created.

The engine will be self-sustaining and does not really require installation though the user will need to know how to specify plugins.

3.4 Constraints

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

3.5 System Model

3.5.1 Scenarios

The engine must be programmed in *C♯* as specified by the client. It will be developed in Microsoft Visual Studio. It will itself be a library (.dll) and should be compatible with modern systems.

3.5.2 Use Case Models

TODO

3.5.2.1 Actors

3.5.2.2 Use Cases

3.5.3 Object Models

TODO

3.5.3.1 Data Dictionary

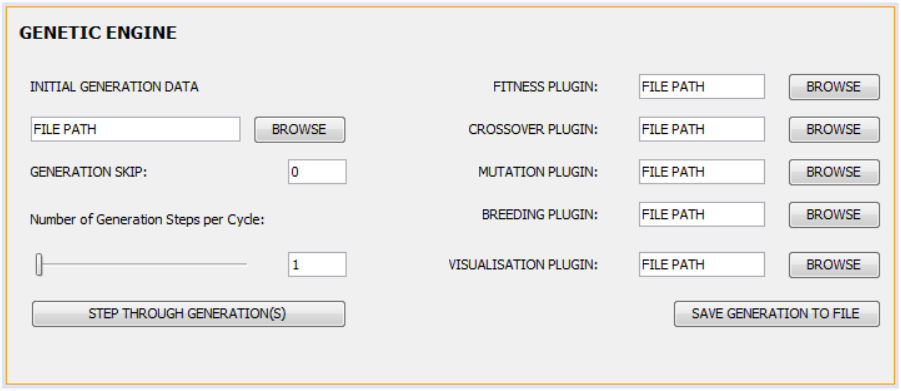
3.5.3.2 Class Diagrams

3.5.4 Dynamic Models

TODO

3.5.5 User Interface - Navigational Paths and Screen Mockups

Figure 1: The screen mock-up of the Genetic Engine Interface. Each plugin is to be specified by their file-path.



4 Glossary

Figure 2: Pathfinding using Genetic Algorithms shown visually in Gridview