SE 3XA3: Module Guide
GrateBox

Team 8, Grate
Kelvin Lin (linkk4)
Eric Chaput (chaputem)
Jin Liu (liu456)

December 8, 2016

Contents

List of Tables

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 06	1.0	Creation of template and first additions
Nov 08	1.1	Added all elements not directly reliant on specific module names/uses
Nov 11	1.2	All but module detail complete
Nov 13	1.3	Final draft and editing complete
Dec 07	1.4	Final Revisions

Note that all changes made for Revision 1 are written in purple.

1 Introduction

The Genetic Cars project seeks to present an accurate depiction of genetic algorithms in action both as a teaching tool and as a source of entertainment. The requirements for this document, outlined in the software requirements specifications document, outline what the Genetic Cars project must do. This module guide will present how it will go about doing so by presenting the various modules of the Genetic Cars project. The Module internal specification (MIS) will specify the exact methods and functions of the application in more detail.

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We are using decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes.

Our design follows the rules laid out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

This MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 2 lists the anticipated and unlikely changes of the software requirements. Section 3 summarizes the module decomposition that was constructed according to the likely changes. Section 4 specifies the connections

between the software requirements and the modules. Section 5 gives a detailed description of the modules. Section 6 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 7 describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 2.1, and unlikely changes are listed in Section 2.2.

2.1 Anticipated Changes

AC1: The changes that may arise from the placement of the Genetic Cars project online (e.g. web hosting choices, website layout, etc.)

AC2: The ability for the user to alter inputs (i.e. the mutation rate, parameters for cars (vertex number etc.), parameters for the environment (gravity etc.))

AC3: Changes to the overall structure of the road (i.e. using different mathematical formulas to generate it, more or less steep overall, length)

AC4: Changes to the aesthetics of the program (i.e. display window shape and size, use of color, etc.)

AC5: Changes to the genetic algorithm (i.e. isolate for fastest car instead of farthest traveled, different means of reproduction, etc.)

AC6: Changes to the initial state of the enviroment and car population (i.e. larger/smaller initial population sizes)

2.2 Unlikely Changes

UC1: Changes to the structure of the vehicle altogether

UC2: Changes to the goal of fitness isolation (i.e. isolating for the worst vehicles instead of the best)

UC3: Changes to the speed of componenets in simulations (i.e. the set speed of the wheels)

UC4: Changes to the library used to generate cars (i.e. changes to the library that result in rounder or more abstract cars)

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware hiding module

M2: Car creation module

M3: Evolve car module

M4: Road creation module

M5: Graphics display module

M6: Genetic Algorithm module

M7: Random seed generation and manipulation module

M8: Fitness determination module

M9: Searching algorithms module

M10: Sorting algorithms module

M11: Population generation algorithms module

Level 1	**Level 2**
Hardware-Hiding Module	M1
Behaviour-Hiding Module	M2 M3 M4 M5 M6
Software Decision Module	M7 M8 M9 M10 M11

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules and anticipated changes and modules is listed in Table 6.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

The system is designed to satisfy the requirements developed in the SRS. Throughout this stage the system is decomposed into modules and analyzed.

5.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

5.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

5.2.1 Car creation module

Secrets: The algorithms that create cars.

Services: Creates a car object according to the values present in the input chromosome (definition, x and y vertex arrays, wheel positions array, wheel radius array, and fitness).

Implemented By: CarObject.js, MakeCar.js

5.2.2 Evolve car module

Secrets: The algorithms that evolve the cars between generations.

Services: Creates the next generation of cars determined by the initial input generation by crossbreeding to create offspring as well as mutating offspring according to to a mutation rate.

Implemented By: GrateBox.js

5.2.3 Road creation module

Secrets: The algorithms that create and govern the road used by the simulation

Services: Generates a randomly created road the becomes progressively steeper on which to simulate the cars.

Implemented By: Path.js

5.2.4 Graphics Display module

Secrets: The algorithms that create the graphical display and elements from all other module products.

Services: Creates a graphical representation of the simulation for the user to see as well as implementing the physics from a library of said simulation.

Implemented By: GrateBox.js

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: Box2D.js

5.3.1 Genetic Algorithm module

Secrets: The algorithms determining how each car generation changes from the last.

Services: Determines which parent cars create the offspring.
Creates the offspring. Mutates the genes in the offspring's chromosomes. Creates final chromosome to be used with the Create car and Evolve car modules.

Implemented By: GrateBox.js

5.3.2 Random seed generation and manipulation module

Secrets: The algorithms determining the random elements of the application.

Services: Generates a random seed to be used for by the Car creation and Road creation modules. Generates random ints and floats for all of the behaviour hiding modules that use them.

Implemented By: GrateBox.js

5.3.3 Fitness determination module

Secrets: The algorithms determining the fitness of a car depending on its performance.

Services: Sets the criteria for score. Measures each car's score to determine which is the highest. Generates final fitness to be used by the Evolve car module.

Implemented By: GrateBox.js

5.3.4 Searching algorithms module

Secrets: The algorithms that search through data (mostly in arrays)

Services: Brute force search methods through arrays for the Create car, Evolve car, and Road creation modules. Uses selection sort for this purpose

Implemented By: GrateBox.js

5.3.5 Sorting algorithms module

Secrets: The algorithms that sort through data (mostly in arrays)

Services: Brute force sort methods through arrays for the Create car, Evolve car, and Road creation modules. Uses selection sort for this purpose

Implemented By: GrateBox.js

5.3.6 Population generation algorithms module

Secrets: The algorithms that effect the generation of the initial population and the proceeding populations.

Services: Generates the initial population with all parameters. Generates the proceeding populations using chromosomes generated in other modules.

Implemented By: GrateBox.js

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes. Requirements are outlined in greater detail in the SRS found here. Design decisions based on requirements have been considered in module decomposition.

Req.	Modules
Req 1: Car body parameters	M2
Req 2: Wheel number parameters	M2
Req 3: Wheel radius parameters	M2
Req 4: Wheel position parameters	M2
Req 5: Min weight parameters	M2
Req 6: Max weight parameters	M2
Req 7: Generation display parameters	M5
Req 8: Fitness display parameters	M5
Req 9: Random seed parameters	M7
Req 10: Mutation rate parameters	M6

Req.	Modules
Req 11: Cars per generation parameters	M3
Req 12: Road generation parameters	M4
Req 13: Min cars per generation parameters	M3
Req 14: Max cars per generation parameters	M3
Req 15: Top cars parameters	M3, M5
Req 16: Max top cars parameters	M3
Req 17: Min top cars parameters	M3
Req 18: Non-moving parameters	M2, M5
Req 19: Fitness parameters	M3
Req 20: Default value replacement parameters	M2, M4

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M5
AC2	M2, M4, M6, M11
AC3	M4
AC4	M5
AC5	M3, M6
AC6	M2, M11

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B.

M2 uses M7, M10 and M9.

M3 uses M6, M8, M10, M9, and M7.

M4 uses M7.

M5 uses M2 and M4.

M1, M6, M7, M8, M10, M9, and M11 are all independent modules.