SE 3XA3: Software Requirements Specification
Title of Project

Team #1, Team Name
Anjola Adewale and adewaa1
Sheridan Fong and fongs7
Chelsea Maramot and maramotc

March 18, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

1 Introduction

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 advocate a 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. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed 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.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The 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

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The format of the initial input data.

AC3: The grammar of python through future releases or the inclusion of older python grammars

AC4: New funtionality of the Pygame library through future releases

AC5: The format of the final packaging and distribution format of the program

AC6: The graphics are likely to change for obstacle and character based on themes and graphical design decisions.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

UC3: The algorithms for the game points calculation and leaderboard display

UC4: The objective of the game which is to score as many points as possible.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 3. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Obstacle Module

M2: Character Module

M3: Cloud Module

M4: Global Variable Module

M5: Large Obstacle Module

M6: Small Obstacle Module

M7: Images Module

M8: Bird Module

M9: Instructions display Module

M10: ChromeDino Module

M11: Leaderboard Module

M12: Leaderboard display Module

M13: Game Settings Display

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 is listed in Table 4.

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

Level 1	**Level 2**
Hardware-Hiding Module	Python, Visual Studio
Behaviour-Hiding Module	Global Variable Module Images Module Instructions Display Module Leaderboard Display Module Game Settings Display
Software Decision Module	ChromeDino Module Obstacle Module Small Obstacle Module Large Obstacle Module Character Module Cloud Module Bird Module Leaderboard Module

Table 2: Module Hierarchy

Level 1	**Level 2**
Model	Obstacle Module Small Obstacle Module Large Obstacle Module Character Module Cloud Module Bird Module Leaderboard Module
View	Global Variable Module Images Module Instructions Display Module Leaderboard Display Module Game Settings Display
Controller	ChromeDino Module

Table 3: Module Hierarchy: MVC Model

the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

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. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules (M13)

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: Python Libraries

5.2 Behaviour-Hiding Module

5.2.1 Global Variable Module (View)

Secrets: The parameters required for the implementation of the game.

Services: This module provides the global variables used to track the status of the display, game, and settings.

Implemented By: Python Libraries

5.2.2 Images Module (View)

Secrets: The images and graphics of the game.

Services: This module provides character, obstacle, and background images for the game interface. It is activated by the controller and the model determines which images should be displayed.

Implemented By: Python Libraries

5.2.3 Instructions Display Module (View)

Secrets: The contents of the instruction page.

Services: This module provides output to the user screen displaying the game's instructions. It is activated by the controller and the model determines if the Instructions Display Module should be used.

Implemented By: Python Libraries

5.2.4 Leaderboard Display Module (View)

Secrets: The contents of the leaderboard display.

Services: This module provides output to the user screen displaying the current leaderboard. It is activated by the controller and the model determines if the Leaderboard Display Module should be used.

Implemented By: Python Libraries

5.2.5 Game Settings Display Module (View)

Secrets: The contents of the required behaviours.

Services: This module provides output to the user screen displaying the available game settings. It is activated by the controller and the model determines if the Game Settings Display Module should be used.

Implemented By: Python Libraries

5.2.6 Leaderboard Calculation Module (Controller)

Secrets: Algorithm for determining leaderboard members.

Services: This module outputs the top scorers and the scores that will be displayed by leader_board display.

Implemented By: Python Libraries

5.3 Software Decision Module

5.3.1 ChromeDino Module (Controller)

Secrets: The algorithm for determining the game logic and page navigation.

Services: Takes the user input and navigates through the game pages and/or game track.

Implemented By: Python Libraries

5.3.2 Obstacle Module (Model)

Secrets: The characteristics of an obstacle.

Services: This module can draw and change the speeds of obstacles on the display and output them to the view. It is responsible for creating the objects.

Implemented By: Python Libraries

5.3.3 Small Obstacle Module (Model)

Secrets: The the y-location and specific small object image.

Services: It is responsible for creating the objects and setting their height on the screen and choosing the specific small object image.

Implemented By: Python Libraries

5.3.4 Large Obstacle Module (Model)

Secrets: The module that sets the y-location and specific large object image.

Services: It is responsible for creating the objects and setting their height on the screen and choosing the specific large object image.

Implemented By: Python Libraries

5.3.5 Character Module (Model)

Secrets: The characteristics of the character such as the positions, images, and jumping velocity.

Services: Changes the image output to the screen based on the user input. The images are displayed based on real world properties such as gravity.

Implemented By: Python Libraries

5.3.6 Cloud Module (Model)

Secrets: The algorithm that generates a random cloud in the game background.

Services: This module updates the position of the cloud in the background and creates a new cloud to display.

Implemented By: Python Libraries

5.3.7 Bird Module (Model)

Secrets: The class that generates a bird object and it's characteristics.

Services: Creates a cloud with random y-axis positioning and draws it to the screen.

Implemented By: Python Libraries

5.3.8 Leaderboard Module (Model)

Secrets: The algorithm that stores and tracks the changes in the leaderboard.

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

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
	Functional Requirements
FR1	M9
FR2	M9
FR3	M11, M12
FR4	M11, M12
FR5	M11, M12
FR6	M13
FR7	M13
FR8	M1, M2, M4, M7, M10, M13
FR9	M1, M2, M4, M7, M10, M13
FR10	M4
FR11	M13
FR12	M9
FR13	M10
FR14	M10, M11
FR15	M10, M11
FR16	M10
FR17	M10
FR18	M10, M11, M12
FR19	M10, M11, M12
	Non-functional Requirements
LF1	M9, M12, M13
LF2	M9, M12, M13
LF3	M9, M12, M13
LF4	M9, M10, M12, M13
UH1	M10
UH2	M10
UH3	M9, M10
UH4	M9, M12, M13
UH5	M1, M2, M3, M5, M6, M7, M8, M9, M10, M13
UH6	M9, M10, M13
UH7	M9
UH8	M9, M12, M13

Req.	**Modules**
	Non-functional Requirements
UH9	M10
UH10	M9, M12, M13
UH12	M9, M12, M13
PE1	M4, M10, M11
PE2	M10
PE3	M4, M10
PE4	M4, M10, M11, M12
PE5	M10, M11, M12
PE6	M10
PE7	M10
PE8	M10, M11, M12
PE9	M1, M2, M3, M5, M6, M7, M12, M13
PE10	M10
PE11	M10, M11, M12, M13
PE12	M10
PE13	M10
PE14	M10
PE15	M10
MA1	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13
MA2	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13
MA3	M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M13
MA4	M10
SR1	M10, M11, M12
SR2	M13
SR3	M10
SR4	M11, M12
SR5	M11, M12
SR6	M10, M11
CP1	M13
CP2	M10

Table 4: Trace Between Requirements and Modules

AC	Modules
AC1	M10
AC2	M10
AC3	M10
AC4	M10
AC5	M9, M12, M13
AC6	M1, M2, M3, M5, M6, M7, M8

Table 5: 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. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

Figure 1: Use hierarchy among modules