SE 3XA3: Software Requirements Specification
Title of Project

Team #, Team Name
Kristine Uchendu (uchenduc)
Jason Nam (namy2)
Rylan Sykes (sykesr)

March 18, 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
March 18, 2022	0	Revision 0

1 Introduction

A key part of the development process is documentation of the modular breakdown of the system. The modular breakdown is exactly what it sounds like the breaking down of our system (the Sketchy Super Mario Bros game) into individual modules that each hold one secret. This document will be outlining the systems existing modules, what secret they are hiding as well as how they are designed for change. In the Anticipated and Unlikely Changes section, described will be changes foreseen in the design of the system as well as changes that appear to be unlikely for the system. In the following sections, the module structure will be discussed along with how they make up the system. Finally, there should be visual aids throughout the document with one of them being a table mapping how the systems requirements and anticipated changes are met by the systems 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 changes that are foreseen in the design process of the system. These changes greatly factor into how the system is decomposed into modules, because the system is decomposed such that these changes are easier to make down the line. Essentially, each change described here should result in the modification of a singular model, should it actually be implemented. This is the design for change approach.

AC1: The addition of new power up items for Mario to interact with.

AC2: The addition of new levels, and worlds (groupings of levels with the final challenge at the end.

AC3: The addition of more moving actors (antagonists for Mario to interact with in the game).

AC4: Changing the behaviours of existing modelled objects in the game.

AC5: The addition of new audio files for sound effects

AC6: Updates may be made to the graphics/animations of the characters.

AC7: The creation of a starting page for the game as well as the creation of end of game pages.

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

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

UC3: The number of players playing one game together (it will likely remain a single player game).

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: Gradle Configuration Module

M2: Launcher Module

M3: Audio Module

M4: Actions Module

M5: Input Module

M6: Models Module

Level 1	**Level 2**
Hardware-Hiding Module	**M5:** Input Module
Behaviour-Hiding Module	**M3:** Audio Module **M4:** Actions Module **M6:** Models Module
Software Decision Module	**M1:** Gradle Configuration Module **M2:** Launcher Module

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 is listed in Table 3.

There are many design decisions that need to be made to realize requirements for this system. The design of the system must make it easy to use and access. The user must be able to execute the system without failure, and perform actions with ease and simplicity. The Gradle configuration module and the launcher module of the system will be designed so that the game is accessible with simple configuration of the system. This will be able to fulfill FR1, NF8, and NF9. The design of the system must also allow user to perform all basic command of actions to Mario within the system. Mario must be able to appropriately move right, left, and up given a input by the user. The input must be delivered by a keyboard input. The design must accept appropriate input, whether that be left, right, or up. The design must not accept any other input that has no correlation within the system. This design decision will meet FR3, FR4, FR6, and NF10. The design must perform all interaction between models properly. One case could be when Mario interacts with a mushroom. Mario must become big Mario. Another case might involve a goomba interacting with Mario. Mario must die or big Mario must shrink if touched by a goomba. These interactions between models must be included in design decisions.

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. 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

5.1.1 Input Module (M5)

Secrets: The data structures and algorithms used for the keyboard and mouse inputs to interface with the software.

Services: Serves as an interface between the keyboard & mouse and system

Implemented By: Mouse & Keyboard

5.2 Behaviour-Hiding Module

5.2.1 Audio Module (M3)

Secrets: The algorithms used to transmit the data stored in the audio files to be played through the users speakers.

Services: Acts as an API between the programmer and the hardware to programatically play sounds and music

Implemented By: libGDX

5.2.2 Actions Module (M4)

Secrets: The behaviour of all actors in the game, including the behaviour of when the user sends an input to control Mario.

Services: Acts as a communication layer between the keyboard & mouse input and the libGDX actor actions

Implemented By: N/A

5.2.3 Models Module (M5)

Secrets: The data structures and algorithms used to store and manipulate the state of each actor in the game

Services: Allows the actors to exist in a game world and communicate with other actors

Implemented By: N/A

5.3 Software Decision Module

5.3.1 Gradle Configuration Module (M1)

Secrets: The configurations used to define how Gradle will interact with the system

Services: Allows Gradle to interface with the game making it possible to install and update package necessary to execute the game.

Implemented By: Gradle

5.3.2 Launcher Module (M2)

Secrets: The data structures used to instantiate the game

Services: Allows the user to launch the game

Implemented By: N/A

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
FR1	M1, M2
FR2	M1, M2, M5
FR3	M4, M5
FR4	M4, M5
FR5	M4, M6
FR6	M4
FR7	M4, M6
FR8	M4, M6
FR9	M3, M4, M6
FR10	M3, M6
FR11	M3, M6
FR12	M3, M4, M6
FR13	M3, M4, M5, M6
FR14	M3, M4, M5, M6
FR15	M1, M2, M5
NF1	M1, M2, M5, M6
NF8	M1, M2
NF9	M1, M2
NF10	M5
NF11	M1, M2
NF12	M1, M2
NF17	M3, M6

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M3, M4, M6
AC2	M3, M6
AC3	M3, M6
AC4	M4, M5, M6
AC5	M3
AC6	M4, M6
AC7	M1, M2, M3, M5, M6

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. 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.

Hierarchy Between Modules.png

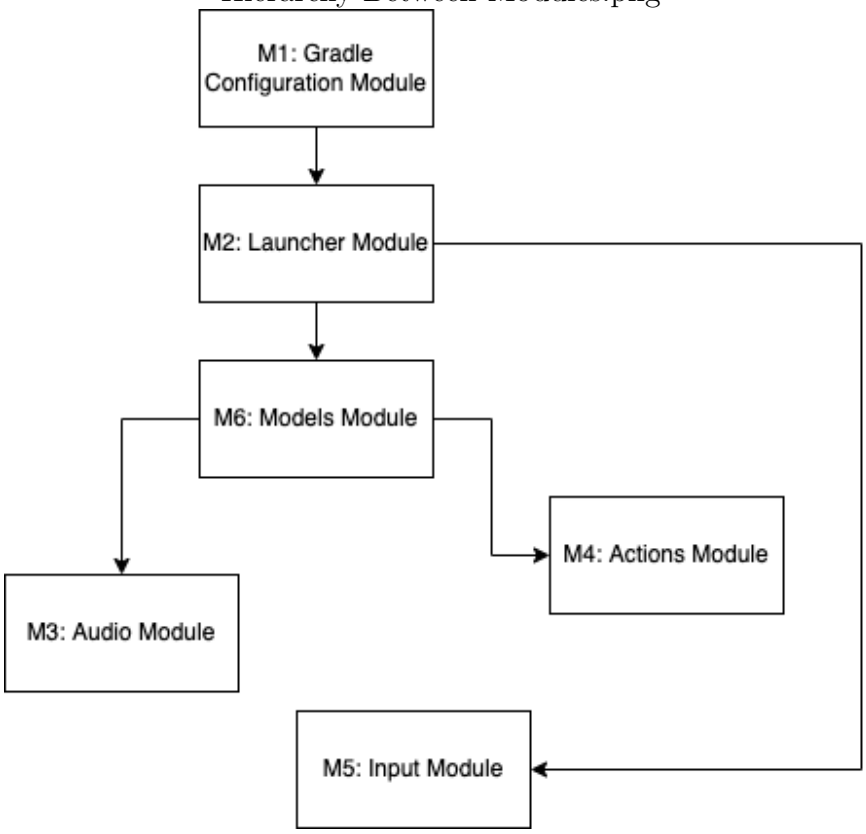

Figure 1: Use hierarchy among modules