SE 3XA3: Software Requirements Specification
Mario Level X

Group 210, Group 210
Edward Liu, liuz150
Connor Czarnuch, czarnucc
Ahmad Gharib, ghariba

March 14, 2020

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2020-03-13	1.0	Revision 0 of the Module Guide

1 Introduction

1.1 Overview

The Mario Level X project is an implementation of an existing open-source software application that adds onto the features of Mario Level 1. Originally allowing the user to play the first level of the original Mario game, Mario Level X expands upon this idea to allow users to modify the game level and save their custom maps.

1.2 Context

The module guide is one of many documents that represents the system. This document, in unison with the other documents submitted specifies all the functional requirements, non-functional requirements for the system (SRS). This document expands upon this by displaying the modular decomposition of the project and how the system is intended to satisfy all the functional and non-functional requirements. The following document, the MIS, will show in detail the modules that are contained in the system, as well as all the attributes for each respective class (environment variables, assumptions, access methods, constants, and variable types)

1.3 Design Principles

Empty - come back to this

1.4 Document Outline

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: All classes that use the Collider class are likely to change in implementation to accommodate for input format changes.

AC2: The player movement will be modified to be more consistent to input commands.

AC3: The format of the input map files will be changed to increase readability.

AC4: The format of the output map files will be changed to increase readability.

AC5: All classes that rely on map files will be modified to account for new map modification features.

AC6: Music implementation will be changed such that it is consistent on all platforms.

AC7: Input/Output devices (Input: Keyboard, Mouse. Output: Screen). Previously the project only required keyboard input, but will be modified to use mouse input for map creation.

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: Already existing screens (main menu, gameplay, death) are expected to remain unchanged.

UC2: The simple state-machine is to remain unchanged.

UC3: Current implementation that loads in sprites into game is to remain unchanged.

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: Behaviour-Hiding Module

M3: Software Decision Module

Level 1	**Level 2**
Hardware-Hiding Module	Python Environment Pygame Module
Behaviour-Hiding Modules	Components Modules Main Menu Module Load Screen Module Level1 Module
Software Decision Modules	Main Module Setup Module Game Sound Module Tools 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.

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 Python Environment

Secrets: The implementation of the Python interpreter.

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

5.1.2 Pygame Module

Secrets: Open source library for developing games in Python.

Services: Allows the Python interpreter to display graphics and register keypresses and mouse clicks. This module provides the interface between the hardware inputs of the computer and the software event triggered by an input.

Implemented By: Python

5.2 Behaviour-Hiding Modules

5.2.1 Components Modules (Mario)

Secrets: Animation behaviour for Mario, movement behaviour, fireball behaviour, powerup behaviour.

Services: This module provides a class that can be implemented in the Software decision modules to affect the Mario sprite and it's actions based on keyboard inputs registered by the Hardware Hiding Modules.

Implemented By: Python

5.2.2 Components Modules (Bricks)

Secrets: Animation behaviour for bricks, powerup behaviour.

Services: This module provides a class for the Software decision modules to implement to direct the behaviour of the bricks when Mario collides with them, breaking them, or releasing powerups.

Implemented By: Python

5.2.3 Main Menu Module

Secrets: Game startup code and main menu layout and actions.

Services: This module provides a class for the Pygame environment to implement that loads the game to an initial main menu game state allowing the player to start the game.

Implemented By: Python

5.2.4 Load Screen Module

Secrets: Loading screen layout and sprites

Services: This module provides a class for the Pygame environment to implement that displays the loading screen after the game is started from the main menu, or if Mario dies. Shows current time, lives left, and score.

Implemented By: Python

5.2.5 Level1 Module

Secrets: Game environment.

Services: This module provides a class for the Pygame environment to implement that loads a level stored as a json file using the tools Module in the Software Decision Module. The class is shared to the Pygame environment via the Main module in Software Decision Modules. All component modules are loaded into this and data read from the JSON file places each entity on the map.

Implemented By: Python

5.3 Software Decision Module

5.3.1 Main Module

Secrets: Implements the classes that hold each game state to allow the user to run the game.

Services: Each class is loaded into a class dictionary and that dictionary is added to a Pygame function which adds all of the states and state transitions to the game's run-time.

Implemented By: Python

5.3.2 Setup Module

Secrets: Graphics Environment.

Services: This module calls Pygame functions to set up the run time window to hold the game's graphics. This module also loads all of the games sounds, fonts, and sprite image files into the game's memory.

Implemented By: Python

5.3.3 Game Sound Module

Secrets: Plays game sounds.

Services: From the sound files that are loaded, this module determines when to play each sound. Such as Background music, jumping sound effects, winning music, etc.

Implemented By: Python

5.3.4 Tools Module

Secrets: Controls all the game's states.

Services: This module provides functions to load all sprites, sounds, music to the game from a file. This module provides the functions and methods to allow the game to transition between states.

Implemented By: Python

6 Traceability Matrix

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

6.1 Functional and Non-Functional Requirements

Req.	**Modules**
FR1	M2, M3
FR2	M2, M3
FR3	M1
FR4	M2, M3
FR5	M2
FR6	M1
NFR1	M2, M3
NFR2	M1, M2, M3
NFR3	M1, M2, M3
NFR4	M2
NFR5	M1
NFR6	M2
NFR7	M2
NFR8	M2
NFR9	M1, M2, M3
NFR10	M3
NFR11	M1
NFR12	M1
NFR13	M3
NFR14	M2, M3
NFR15	M1, M3

Table 3: Trace Between Requirements and Modules

6.2 Anticipated Changes

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

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

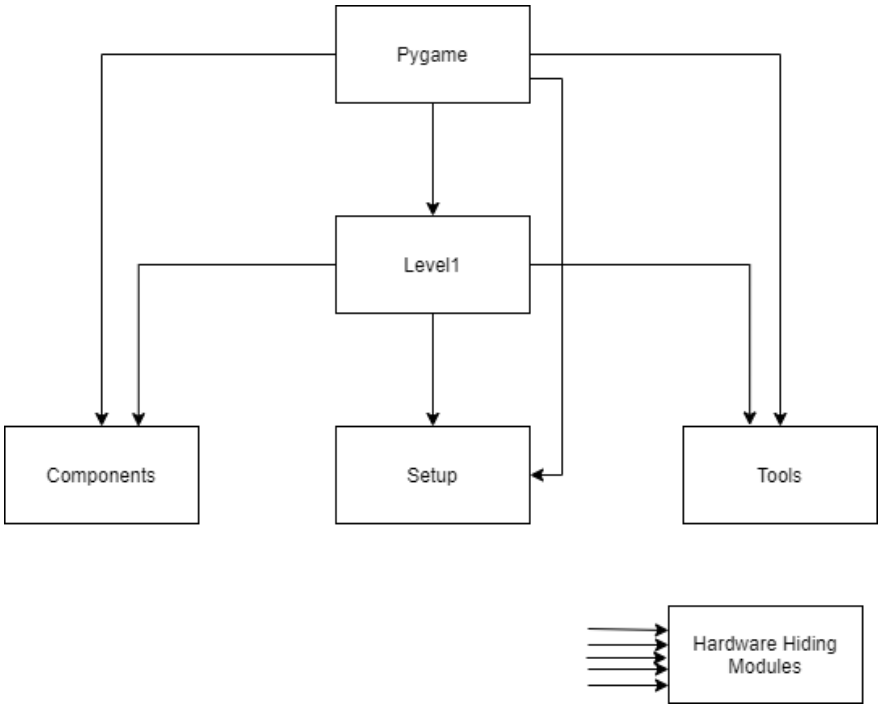

Figure 1: Use hierarchy among modules