# SE 3XA3: Software Requirements Specification
# Legend of Python

Group # 1, Lava Boys Inc.
Bilal Jaffry, affryb
Giacomo Loparco, loparcog
Lucas Zacharewicz, zacharel

December 3, 2018

# Contents



# List of Tables

# List of Figures



Table 1: **Revision History**

| Date | Version | Notes |
|---|---|---|
| November 7th | 1.0 | Started the Document |
| November 9th | 1.1 | Finished the Document |
| December 3rd | 1.2 | Rev 1 Update |

# 1 Introduction

## 1.1 Overview

The Legend of Python projects purpose is to create a re-implementation of a open-source project for the original The Legend of Zelda, released in 1986 for the Nintendo Entertainment System video-game console. This project will allow a user to control the player character, while exploring several designed levels.

## 1.2 Context

This Module Guide's purpose is to provide a modular decomposition of the Legend of Python project. This document also provides a look into the modular structure of the project and how each of these modules collectively meet the functional and non-functional requirements for system, outlined in the Software Requirements Specification. This document is supplemented by the presence of the Module Interface Specification, providing information for all the program syntax and the semantics regarding each module and its respective methods.

## 1.3 Design Principles

The design principles utilized in the development of this project and displayed in this Module Guide are the properties of Encapsulation, Information Hiding and that of the Uses Relation Hierarchy. The Uses Relation Hierarchy should have no cycles, ensuring efficient and independent software design. Module's that both rely on each other should not be present, and the system should follow the low coupling and high cohesion principle. This principle would ensure the system and its module components are related strongly. Information Hiding for the project hides the process in how each module works in the system when acted upon by the user. Finally, the principle of Encapsulation is to be followed allowing for changes in the implementation of a module but not the interface the module is used in.

## 1.4 Document Structure

The following Module Guide is as follows:

- **2** Anticipated and Unlikely Changes affecting the system.
- **3** Module Hierarchy, detailing all the modules and the hierarchy, categorizing each module in their respective categories of Hardware/Behaviour/Software hiding.
- **4** Connection Between Requirements and Design, detailing how each module satisfies the requirements of the software.
- **5** Module Decomposition, providing the module, which component of Hardware/Behaviour/Software hiding it is a member of, and what function each module accomplishes in the entire system.
- **6** Traceability Matrix, providing the association of the requirements of the software system to the modules implemented, as well as the matrix representing the trace back for modules affected by the the anticipated changes.
- **7** Uses Hierarchy, providing the uses relations between each of the modules.

# 2 Anticipated and Unlikely Changes

## 2.1 Anticipated Changes

This section will display the likely changes the software system will encounter during its use and distribution.

**AC1:** The data structure implementation of the various levels of the system.

**AC2:** Structure of display elements in User Interface through the Heads Up Display.

**AC3:** Basic NPC enemy placement in individual rooms.

**AC4:** Level transitions collision trigger for user movement.

**AC5:** Font rendering class will change its inheritance from Pygame Sprite object to Font object.

### 2.2 Unlikely Changes

**UC1:** Input/Output devices to control the user player. The system will only respond to keyboard input and output using the available display devices.

**UC2:** The height/width for the output window and Heads Up Display.

**UC3:** The NPC enemy movement and attack logic .

**UC4:** Spritesheets for the player, enemy, items, and dungeon.

## 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:** Aquamentus

**M2:** Boomerang

**M3:** Boss

**M4:** Constants data

**M5:** Enemy

**M6:** Fireball

**M7:** Health Bar

**M8:** Item

**M9:** Keese

**M10:** Player

**M11:** Render Font

**M12:** Rupee

**M13:** Sprite Sheet

**M14:** Stalfos

**M15:** Sword

**M16:** Door

**M17:** Level

**M18:** Level Data

**M19:** Level Manager

**M20:** Wall

**M21:** Colour Data

**M22:** Window Data

**M23:** Main Run File

**M24:** Block

**M25:** Key

# 4 Connection Between Requirements and Design

The design of the system was designed to satisfy all the requirements specified in the SRS (Software Requirements Specification). The python module Main Run File is the class that consolidates all the components of the game system is the file ran when using the custom makefile for the system. The Player module of the system controls the player character movement, satisfying with the Look and Feel/Usability requirements of the system, ensuring that the movement and attack animations are accurate to the original release. The controllable player character responds accurately and consistently to users keyboard inputs.The input parameters of the system are also satisfied, with the [W,A,S,D] keyboard keys initialized for movement of the player character. The system does not interfere with any other processes running on the given

| **Level 1** | **Level 2** |
|---|---|
| 70.3Behaviour-Hiding Module | Aquamentus |
| | Boomerang |
| | Fireball |
| | Health Bar |
| | Item |
| | Keese |
| | Player |
| | Rupee |
| | Stalfos |
| | Sword |
| | Door |
| | Level |
| | Level Manager |
| | Wall |
| | Block |
| 30.3Software Decision Module | Boss |
| | Colour Data |
| | Constants data |
| | Enemy |
| | Level Data |
| | Main Run File |
| | Render Font |
| | Sprite Sheet |
| | Window Data |

Table 2: Module Hierarchy

system, as well as ensuring the system interacts with the respective Operating System properly.

The Sprite Sheet module and the accurately renders sprites to screen without unforeseen rendering issues, satisfying the Performance Requirements of the system, ensuring the various sprites are render precisely and consistently at a 60 Hz refresh rat. The eventual collective file size further supports the

Performance Requirements of the software project ensuring the collective file size of the system is under the 500MB limit. In order to effectively design the software system to ensure the Maintainability and Support Requirements were met, the file paths for the various file used in the system, such as sprites in the Sprite Sheet module and the fonts in the Render Font module, the system was designed to run with local directory based on the Python OS library. This library guaranteed the software can be compiled on various Operating Systems (Windows/Linux) without any possible issues.

The resultant software is kept simple and concise for users to interact with, allowing for users to access and modify this open source software with ease. The design of the software reliably follows all aspects of the original, both culturally and legally, ensuring no Cultural and Legal requirements are violated. The design of the system does not compromise the users data during or after run-time of the software ensuring the Health and Safety Requirements of the system are met consistently.

# 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.

## 5.1 Behaviour-Hiding Module

**Secrets:** Behaviours

**Services:** Describes the visible behaviour of the game. Acts as the interpreter between the Hardware Hiding Module and the Software Decision Module.

**Implemented By:** N/A

### 5.1.1 Aquamentus Module

**Secrets:** How Aquamentus moves and attacks.

**Services:** Controls the boss enemy Aquamentus in the game.

**Implemented By:** Aquamentus

### 5.1.2 Block

**Secrets:** Block creation and collision detection

**Services:** Provides collectible blocks within dungeon levels, accepting any argument of sprite

**Implemented By:** Block

### 5.1.3 Boomerang

**Secrets:** Boomerang movement and hitbox.

**Services:** Controls the movement of the in game item boomerang.

**Implemented By:** Boomerang

### 5.1.4 Fireball

**Secrets:** Fireball movement and hitbox.

**Services:** Controls the fireball projectiles in the game.

**Implemented By:** Fireball

### 5.1.5 Health Bar

**Secrets:** Health bar rendering.

**Services:** Provides the in game health bar for the player.

**Implemented By:** Health Bar

#### 5.1.6 Item

**Secrets:** Item creation, rendering, and behaviour.

**Services:** Provides the in game collectible items.

**Implemented By:** Item

#### 5.1.7 Keese

**Secrets:** Keese movement and animation.

**Services:** Controls the Keese enemies behaviour in the game.

**Implemented By:** Keese

#### 5.1.8 Player

**Secrets:** Player's actions and animations.

**Services:** Provides the player movement and actions for controlling the character in game.

**Implemented By:** Player

#### 5.1.9 Rupee

**Secrets:** Rupee rendering and behaviour.

**Services:** Provides the rupee item in the game

**Implemented By:** Rupee

#### 5.1.10 Stalfos

**Secrets:** Stalfos movement and animation.

**Services:** Controls the Stalfos enemies in the game.

**Implemented By:** Stalfos

#### 5.1.11 Sword

**Secrets:** Sword collision and actions.

**Services:** Provides the sword attack for the player.

**Implemented By:** Sword

#### 5.1.12 Door

**Secrets:** Door behaviour and rendering.

**Services:** Allows for the use of doors to traverse the map.

**Implemented By:** Door

#### 5.1.13 Level

**Secrets:** Level creation and rendering.

**Services:** Allows for rooms on the map to be created.

**Implemented By:** Level

#### 5.1.14 Level Manager

**Secrets:** Map creation and transition behaviour.

**Services:** Allows for rooms to be linked together to make full maps.

**Implemented By:** Level Manager

#### 5.1.15 Wall

**Secrets:** Wall creation and collision detection.

**Services:** Provides collidable walls in the game.

**Implemented By:** Wall

## 5.2 Software Decision Module

**Secrets:** Data structures.

**Services:** The base data structures that model the game's back end.

**Implemented By:** Pygame

### 5.2.1 Boss

**Secrets:** Boss framework and initialization.

**Services:** Provides a bare bones boss game loop to build upon

**Implemented By:** Boss

### 5.2.2 Colour Data

**Secrets:** Colour constants.

**Services:** Provides colour constants for the game.

**Implemented By:** Colour Data

### 5.2.3 Constants Data

**Secrets:** Constants Data.

**Services:** Provides the constants for the majority of game variables.

**Implemented By:** Constants Data

### 5.2.4 Enemy

**Secrets:** Enemy initialization and framework.

**Services:** Provides a bare bones implementation of an enemy game loop

**Implemented By:** Enemy

#### 5.2.5 Level Data

**Secrets:** Data for level creation.

**Services:** Provides the level data for all the rooms on the map.

**Implemented By:** Level Data

#### 5.2.6 Main Run File

**Secrets:** Main game loop.

**Services:** Main game loop, running other modules and producing sounds for the game.

**Implemented By:** Main Run File

#### 5.2.7 Render Font

**Secrets:** How font is rendered.

**Services:** Renders onscreen font.

**Implemented By:** Pygame, Render Font

#### 5.2.8 Sprite Sheet

**Secrets:** Sprite Sheet parsing.

**Services:** Provides spritesheet image grabbing for onscreen graphics

**Implemented By:** Pygame, Sprite Sheet

#### 5.2.9 Window Data

**Secrets:** Window constants.

**Services:** Provides the constants for the window of the game.

**Implemented By:** Window Data

# 6 Traceability Matrix

| F Req. | Modules |
|---|---|
| FR1 | M23 |
| FR2 | M23 |
| FR3 | M23 |
| FR4 | M23 |
| FR5 | M18, M19 |
| FR6 | M18, M19 |
| FR7 | M20, M24 |
| FR8 | M8, M10 |
| FR9 | M8 |
| FR10 | M2, M10 |
| FR11 | M12, M25 |
| FR12 | M4, M10 |
| FR13 | M10, M23 |
| FR14 | M1, M9, M14 |
| FR15 | M1, M3, M5, M9, M14 |
| FR16 | M4, M9, M14 |
| FR17 | M1, M3, M5, M9, M10, M14 |
| FR18 | M3, M5, M10 |
| FR19 | M10, M23 |
| FR20 | M8, M9, M14, M19 |
| FR21 | M8 |
| FR22 | M8, M10 |
| FR23 | M8, M10 |
| FR24 | M10, M16 |
| FR25 | M23 |
| FR26 | M8, M10, M23 |
| FR27 | M8, M10, M18, M19 |

Table 3: Trace Between Functional Requirements and Modules

| N-F Req. | Modules |
|---|---|
| NFR1 | M1, M3, M5, M9, M10, M14, M19, M23 |
| NFR2 | M23 |
| NFR3 | M22, M23 |
| NFR4 | M19, M23 |
| NFR5 | M1, M2, M9, M19, M13, M14, M15, M16, M17, M20 |
| NFR6 | M10, M23 |
| NFR7 | M23 |
| NFR8 | N/A |
| NFR9 | M* |
| NFR10 | M10, M23 |
| NFR11 | M22, M23 |
| NFR12 | N/A |
| NFR13 | N/A |
| NFR14 | N/A |
| NFR15 | N/A |
| NFR16 | N/A |
| NFR17 | N/A |
| NFR18 | N/A |
| NFR19 | M13, M19 |
| NFR20 | N/A |
| NFR21 | N/A |
| NFR22 | N/A |
| NFR23 | N/A |
| NFR24 | N/A |

Table 4: Trace Between Non-Functional Requirements and Modules

| AC | Modules |
|---|---|
| AC1 | M18, M19 |
| AC2 | M7, M11, M12 |
| AC3 | M2, M3, M5 |
| AC4 | M1, M3, M18 |
| AC5 | M18 |
| AC6 | M10, M16, M19 |
| AC7 | M11 |

Table 5: Trace Between Anticipated Changes and Modules

# 7 Use Hierarchy Between Modules

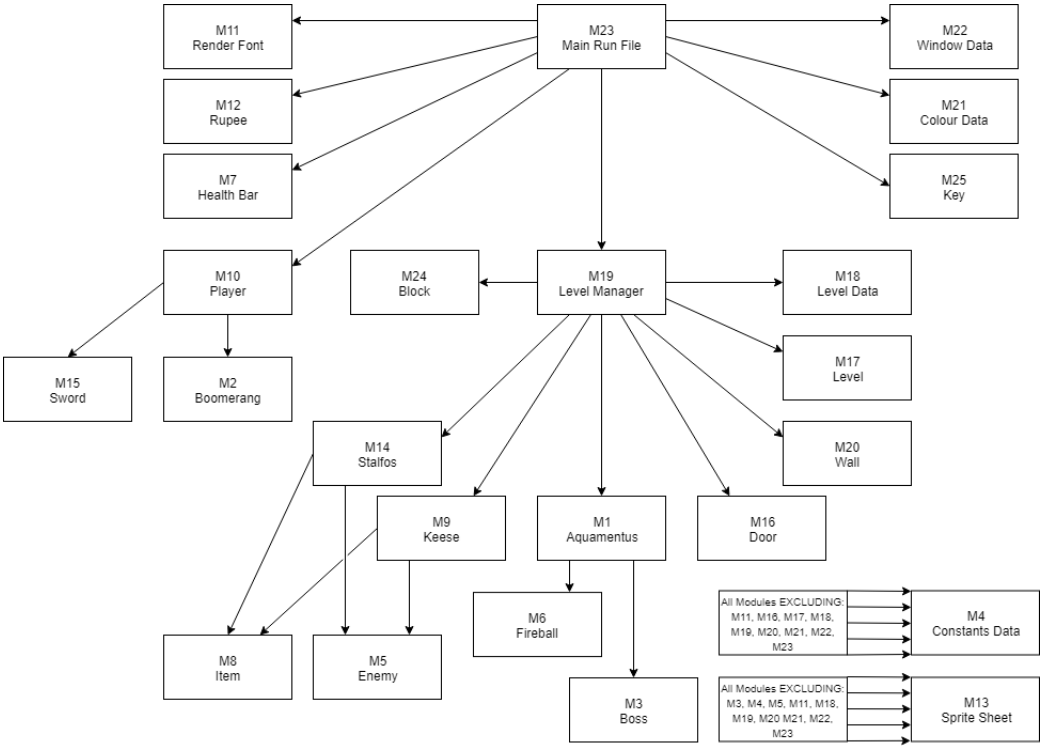


Figure 1: Use hierarchy among modules