SE 3XA3: ~~Software Requirements Specification~~
Module Guide
Spiritual Jumper

Team 25, N.L.E.
Ruoyuan Liu, liur19
Zihao Chen, chenz87
Gundeep Kanwal, kanwalg

December 7, 2017

Contents

List of Tables

List of Figures

Table 1: ~~**Revision** History~~

Date	~~**Version**~~ **Developer(s)**	~~**Notes**~~ **Change**
~~Nov.10~~ 11/10/2017	~~1.0~~ Ruoyuan, Zihao, Gundeep	Document created
~~Nov.10~~ 12/06/2017	~~1.1~~ Ruoyuan, Zihao, Gundeep	~~Document Rev 0 finished~~ Document Rev 1 Finished

1 Introduction

1.1 Project Overview

The Spiritual Jumper ~~Project~~ aims at recreating and enhancing the gameplay of an classic game named "Doodle Jump", based on an existing project on Github. ~~Other than re-implement the source code. The N.L.E team is planning to complete a series of formal documentation and testing to deliver a more comprehensive and complete project to the public.~~ The motivation behind this document is to allow for the decomposition of systems that plan to scale into the future with updates, which allow for developers to concurrently develop components without risk of components not working with each other. This project is concerned with creating a modular system design for the game that is well documented to allow for future upgrades to improve on current game mechanics. Currently the system design principle is that of Model-View-Controller (MVC software design pattern) which is believed to be the best fit for allowing Spiritual Jumper to be modular.

1.2 Document Context

This Module Guide ~~Document~~ is used to comprehensively describe the different modules in the Spiritual Jumper project, also provide a way to overlook the project design as a whole from conceptual perspective.

The MG(Module ~~Document~~ Guide) is based on the previous SRS(Software Requirements Specification) document. The SRS describes the system the MG should be able to accomplish, also with potential changes the MG should cover.

The MIS(Module Interface Specification) should be complete after MG is finished, which com-ntains actual implementation details of the modules that mentioned in MG.

1.3 Design Principle

In order to keep the project as robust and maintanable as we could, the N.L.E team is obligated to implement the following design principle: information hiding, low coupling and high cohesion.

High cohesion and low coupling make sure that the modules keeps certain independence, which benefits further maintenance and testing. Information hiding ensures that the interface will not be affected.

1.4 Document Structure

1. Overview of the Document.
2. List of changes
3. Hierarchy of current modules
4. Connection between requirements and design
5. Module docomposition details. 6. Relationships between modules, requirements and anticipated changes. 7. Revision history

2 Anticipated and Unlikely Changes

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 format of the initial input data.

AC2: The interface of menu, in game background, buttons etc.

AC3: The character apperance, game appearance and animation. Alse resolution of the game.

AC4: The item function and modification of the game.

AC5: The difficulty of the game in terms of platform generation, monster generation etc.

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 specific hardware on which the software is running.

UC4: The primary structure of the whole 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: Output Window Module

M2: User Input Module

M3: Physics Module

M4: Controller Module

M5: Stat-Record Module

Level 1	**Level 2**
Hardware-Hiding Module	Controller Module
Behaviour-Hiding Module	User Input Module Output Window Module Stat-Record Module
Software Decision Module	Physics 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

Module decomposition is the process of creating a set of sub-modules that adhere to the principle of "information hiding" proposed by Parnas. This principle of information hiding will be used as a fundamental concept for Spiritual Jumper's object oriented development. 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

Module 1

Secrets: How the game is displayed, how the game is run, how the game is closed.

Services: Creates and runs all the states for the program, displays the graphics and all the images and sounds for the user.

Implemented By: GameShell Module

5.2 Behaviour-Hiding Modules

Module 2

Secrets: How Character objects are created and how they move and collide.

Services: Creates a Character ADT with abilities to create them with a number of parameters including their positions and size. Also provides the ability to check if characters are colliding.

Implemented By: Character Module

Module 3

Secrets: How the user enters and saves their names, along with their high scores.

Services: Creates a person object with two inputs, one being the score that the user had received, the other being the name they had added.

Implemented By: ~~person~~ Person Module

Module 4

Secrets: How platforms are created, how they are moved.

Services: Creates the game platforms through the use of a platform object with multiple input parameters dictation it's position and attributes.

Implemented By: ~~platform~~ Platform Module

Module 5

Secrets: How the monsters are created, how they move.

Services: Creates the monsters within the game through the use of a monster object with many parameters within the constructor detailing its position and size.

Implemented By: ~~monster~~ Monster Module

Module 6

Secrets: How the bullets are created, how they move around the game display.

Services: Creates the bullet objects with paramaeters detailing their initial location and size. Controls the movements of the bullets.

Implemented By: Bullet Module

Module 7

Secrets: How the Doodle is created, how it moves around, how it changes directions of its face.

Services: Creates a Doodle object with parameters regarding it's position and size. Controls the movements of the doodle along with the directions of the face.

Implemented By: Doodle Module

5.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are 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: –

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
~~R1~~FR1	~~M1~~ M5
~~R2~~FR2	~~M2 M4~~M5
~~R3~~FR3	~~M2 M5~~M2, M3, M4
~~R4~~FR4	~~M2, M7~~M2
~~R5~~FR5	~~M2, M7~~M3
~~R6~~FR6	~~M1, M3~~M3
~~R7~~FR7	~~M1, M2, M4, M5, M6, M7~~M1

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	~~M1~~M2
AC2	M1
AC3	M2
AC4	M3, M4
AC5	M1, M3

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.

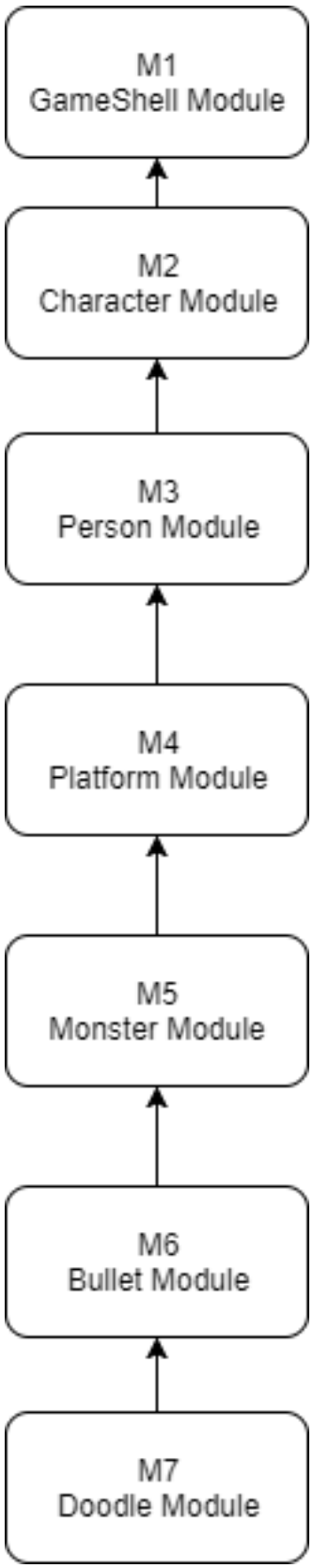

Figure 1: Use hierarchy among modules

The uses hierarchy listed above is read as such: where 1.. 7 are numbered states in a directed acyclic graph and all numbered states are followed by the prior numbered state. Hence 1 to 2 to 3...(where "to" implies the next state) and so forth.