SE 3XA3: Test Plan
Spiritual Jumper

Team 31, N.L.E.
Ruoyuan Liu, liur19
Zihao Chen, chenz87
Gundeep Kanwal, kanwalg

October 27, 2017

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
Date 1	1.0	Test Plan created

This document is a specific test plan based on the project "Spiritual Jumper" by N.L.E team.

1 General Information

1.1 Purpose

This document is for outlining all the testing procedures of the Spiritual Jumper project, and verification if the project meets the functional and non-functional requirements. Readers of this document should be able to develop a comprehensive understanding about the program testing process.

1.2 Scope

This document will list all the technical and conceptual aspect of the program testing procedure, including testing tools , team management, and specific testing process etc.. It will also give out a schedule planned for testing and terminologies and symbols that is used in this document.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
GUI	Graphic user interface
UI	User interface
PoC	Point of concept

1.4 Overview of Document

There are 6 main parts in the documentation.

1. Overview of testing environment, tools, schedule and team coordination.
2. Systematic testing description based on functional and non-functional requirements.

Table 3: **Table of Definitions**

Term	**Definition**
Highest Score Board	A function in the game which users can store and review their highest score.
Jungle mode	A game mode to which the user can change, it will activate new background and music for the game
Menu State	The state when the game is in menu.
Gameplay State	The state that the game is running.

3. Testing about functions based on PoC.
4. Comparison to the original implementation.
5. Testing plan about internal functions and output files.
6. Appendix of symbolic parameters and other additional possible objectives of this project.

2 Plan

2.1 Software Description

Spiritual Jumper is a game which was created in Java. It is based on the game doodle jumper. It takes keyboard and mouse inputs and it outputs the game's graphics which are displayed, along with the state of the game. The mouse input is used to let the user start the game, or to see the high scores. It is also with the keyboard to direct the location of where the user wishes to shoot a projectile. The keyboard input will let the user move side to side, jump and shoot. The functions that are to be tested are the mousePressed, keyPressed, readScores, updateMonster, move, checkDoodleGameOver, Platform, and run function. The "mousePressed" function handles the user input created by the mouse. The "keyPressed" function handles the user input created by the keyboard. The "readScores" function handles the highscores for the game. The "updateMonster" function handles with the creation and movement of the monster. The "move" function handles with the movements of the doodle which the user controls. The "checkDoodleGameOver" function is used to end the game when a user loses. The "Platform" function handles the construction of the platforms used within the game. The "run"

function is used to keep the game running.

2.2 Test Team

The test team for Spiritual Jumper will be the memebers of Group 25, Gundeep Kanwal,Ruoyuan Liu and Zihao Chen.

2.3 Automated Testing Approach

In terms of the automated testing approach, Group 25 will be using JUnit. This was decided as all group members have prior experience using JUnit, and have found it easy to use when working with java. The group plans on using JUnit for testing the "readScores", "keyPressed", "mousePressed" and "checkDoodleGameOver" functions in order to ensure their outputs match up with the expected outputs given certain inputs.

2.4 Testing Tools

As previously stated, JUnit will be used.

2.5 Testing Schedule

As indicated in the teams gantt chart, the testing is scheduled to be completed before the rev0 demonstration.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Area of Testing1

User Input Testing

1. Mouse Testing

   Type: Dynamic

   Initial State: Active Menu State

Input: Mouse Location

Output: Co-ordinates of the Mouse Location

How test will be performed: The game will begin in the Active Menu State. The mouse will be moved to the a point on the menu display to a location of which the co-ordinates are known. The location of the mouse and the co-ordinates are compared to test the accuracy of the mouse input.

2. Keyboard Testing

Type: Functional

Initial State: Active Game State

Input: Keyboard Input

Output: Movement of the player doodle

How test will be performed: The keyboard will be used within the game state to move the doodle a certain way. If the keyboard inputs cause the movement of the doodle to match with the expected movements, the keyboard inputs are validated.

3.1.2 Area of Testing2

Menu and Score Testing

1. Menu to Game Testing

Type: Functional

Initial State: Active Menu State

Input: Mouse Input

Output: State Change to Gameplay State.

How test will be performed: The game will begin in the Active Menu State, and the menu display will be used to navigate the mouse to the play button. The mouse will hover over the play button, modifying the menu display to include a red circle around the play button, and then when the mouse input clicks on the play button, the game will change to the Gameplay State.

2. Menu to Scores Testing

   Type: Functional

   Initial State: Active Menu State

   Input: Mouse Input

   Output: State Change to Scores State.

   How test will be performed: The game will begin in the Active Menu State, and the menu display will be used to navigate the mouse to the scores button. The mouse will hover over the scores button, modifying the menu display to include a red circle around the scores button, and then when the mouse input clicks on the scores button, the game will change to the scores State.

3. Score Testing

   Type: Dynamic

   Initial State: Active Gameplay State

   Input: Score Input

   Output: Score Board

   How test will be performed: The game will end with a user reaching a high score. The user then proceeds to enter their score along with their name which is added to the scoreboard in its correct location. If the score is added to the correct position, the scoreboard is validated.

3.1.3 Area of Testing3

Game Testing

1. Run Testing

   Type: Functional

   Initial State: Active Gameplay State

   Input: Gameplay State

   Output: No Change to state of the Game

How test will be performed: The game will start in the Gameplay state as the user plays the game. The user will proceed to play the game. The game must not end and must remain in the Gameplay state when the user does not meet any of the conditions required for loseing the game. This ensures that the game is running correctly.

2. End Testing

   Type: Functional

   Initial State: Active Gameplay State

   Input: Gameplay State

   Output: State Change to Menu State

   How test will be performed: The game will start in the Gameplay state as the user plays the game. The user will proceed to play the game until they meet on of the conditions required to lose the game. The Gameplay state must conclude and tranfer states to the Menu State.

3. Reset Testing

   Type: Functional

   Initial State: Active Gameplay State

   Input: State Change to Menu State

   Output: Game variables are reset

   How test will be performed: The game will start in the Gameplay state as the user plays the game. The user will proceed to play the game until they meet on of the conditions required to lose the game. The variables used for the game must be reset to their initial values in order for the game to be able to be played again from the starting position.

4. Generating Platform Testing

   Type: Functional

   Initial State: Active Gameplay State

   Input: Start of the Gameplay State

Output: Platforms which are displayed

How test will be performed: The game will start in the Gameplay state as the user plays the game. Platforms which are used for the game must be displayed randomly in a way that the user can use to play the game, and must be continued to be displayed as the user moves up the game as they move from the START_LEVEL onwards.

...

3.2 Tests for Nonfunctional Requirements

3.2.1 Area of Testing1

General Test

1. Usability Testing

   Type: Manual

   Initial State: Active Gameplay State

   Input/Condition: Mouse Movements and Keyboard Inputs

   Output/Result: None

   How test will be performed: The game will be played and checked for how easy it was to play the game; How quick someone can get a hang of the game. This will be done by having group members play the game through while envisioning themselves as someone who does not know how to play the game.

2. Legal Testing

   Type: Manual

   Initial State: Active Gameplay State

   Input/Condition: Mouse Movements and Keyboard Inputs

   Output/Result: None

   How test will be performed: The game will be played and checked to see if it has any legal issues which might be encountered.

3. Legal Testing

   Type: Manual

   Initial State: Active Gameplay State

   Input/Condition: Mouse Movements and Keyboard Inputs

   Output/Result: None

   How test will be performed: The game will be played and checked to see if it has any legal issues which might be encountered.

4. Portability Testing

   Type: Manual

   Initial State: Active Menu State

   Input/Condition: Mouse Movements and Keyboard Inputs

   Output/Result: None

   How test will be performed: The game will be opened on different operating systems to prove it's ability to be portable.

3.3 Traceability Between Test Cases and Requirements

4 Tests for Proof of Concept

4.1 Basic Gameplay

Basic Concept Test

1. Game Start

   Type: Functional, Manual

   Initial State: Home Menu

   Input: Mouse click on "Play Game"

   Output: The program runs successfully.

   How test will be performed: Tester will mouse click the option, "Play Game" from home menu and observe if the program runs successfully.

2. Game Pause

   Type: Functional, Dynamic, Manual

   Initial State: Game On-going

   Input: Mouse click on the "pause" button

   Output: The program will pause the process and display the pause menu.

   How test will be performed: Tester will mouse click the pause button during an on-going game and observe if the program pause its process and display pause menu.

3. Select Mode

   Type: Functional, Static, Manual

   Initial State: Home Menu

   Input:Press F3 to enter "Jungle mode"

   Output: The program will read the input and directed to the appointed state.

   How test will be performed: Tester will click F3 on the keyboard and enter the game and observe if the program is in "Jungle Mode".

4.2 Non-gameplay function test

1. Leaderboard

   Type: Functional, Static, Manual

   Initial State: Home Menu

   Input: Mouse click on "Leaderboard"

   Output: The program will read the input and display the list of history top-10 score and name.

   How test will be performed: Tester will mouse click "Leaderboard" from the home menu and observe if the program display the list of history top-10 score and name.

...

5 Comparison to Existing Implementation

When comparing our implementation of the orignal source code, there are a number of signficant difference between Spiritual Jumper and Doodle Jump. However, the majority of the methods and functions from the orignal are being used with the copy. Due to their, the requirments are similar. Despite this, some notable difference include the difference in music between the two games and our complete implementation of the scoreboard.

6 Unit Testing Plan

The original game of Spiritual Jumper has no built-in testing framework to it and the N.L.E. team will manually write most of the unit testing drivers. Stubs will not be needed during unit testing process. However, adding or disabling some function will be used during testing process in order to reduce the uncertainty of the game itself.

6.1 Unit testing of internal functions

Given the source code of this project is relatively huge, to completely cover all the internal functions would be unrealistic for N.L.E. team. We decided to test most of the crucial parts that have direct or great impact on the game function only.
Platform Generation Functions
The platform generation in this game is the most important part, as it would directly impact the difficulty and playability. That algorithm in the original program would sometimes cause the game unplayable because the character jumped too short compare to the next platform the game generates. In this case, a boundary for platform generation will be introduced to the program and the verification of this function could be fairly easier.
Keyboard Input Functions
The keyboard input function the program uses is a built-in function of Java, which provides the verification of the function.
Monster Functions
The hitbox of the monster in the game is huge and ambiguous. Given that the metrics for the monster can be controlled and constrained easily, the unit testing would mainly be in game. For instance, generating monsters

in a specific location and testing the hitbox with continuously generate new characters jump over it or shoot it.

6.2 Unit testing of output files

The only output file the program generates is the score.txt, which contains data of user's highest score. This function can be easily unit tested by verifying the content of the .txt file after we input a manually written file that would write to the function.

7 Appendix

This section will be revisited once addtional information is needed.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

START_LEVEL = 0.