Test Plan

Gill, Surinder	Hu, Joshua	Lago, Nick
1308896	1311940	1302613

October 21, 2015

1   Revision History

Table 1: Revision History: Test Plan

October 21, 2015	DEVELOPER	CHANGE	REVISION
October 21, 2015	Gill, Surinder	Sections: 2	0
October 21, 2015	Hu, Joshua	Sections: 3	0
October 21, 2015	Lago, Nick	Sections: 1, 4	0

Contents

List of Figures

List of Tables

2 Introduction

2.1 Purpose

The function of our program is to run a 2D survival arcade style game. This game will have to play sounds, render graphics and interpret user inputs. The testing for this software will include the Jasmine Framework as our automated unit testing. For structural testing our group will implement the Jasmine Framework to test collision detection, user input recognition, and cookie storing. Functional testing will consist of sound output testing along with sprite rendering and frame resizing. Mutation testing will also occur on our collision detection function as we alter the parameters for optimal sensitivity and to check to make sure that no other function is altered in the process. Fault testing will occur throughout the design and implementation of this project.

2.2 Environment and Pretest Background

Since this is a brand new project there has been no previous testing. Since this is similar to a popular game called Flappy Bird we do have expected outputs for all the testing that will occur.

2.3 Test Objectives

Testing will allow our group to make sure that the full functionality of our game is being met. Before releasing our software all of our tests must be thoroughly completed and passed. This grants our team the peace of mind that all functionality of our program is working properly.

2.4 Expected Defect Rates

We estimate that no defects will take place during the testing and no further testing will be needed after this set.

2.5 References

There is no previously published documentation on this project. There is documentation however on related documents called "Flappy Bird" which should be more than enough: `http://flappybird.wikia.com/wiki/Category:Template_documentation`

3   Plan

3.1   Software Description

Inputs	Outputs	Functions of Software being tested
Keyboard Input – The game will be relying on keyboard inputs for the two player mode that will be implemented	Server host – The game code will be outputted to a server which will host our website and can be accessed through the world wide web from a display	Cookie Function – This function will hold memory of recent high scores of previous players. It will allow the user to close the game from the browser and access previous scores when reopened if it allows cookies.
Mouse Input – The game will be relying on mouse input for the one player game mode		Sound Function – This function will be importing all the sounds required for the game and then set channels for them to play.
		Window Setter (output) Function - This function looks for different types of devices and has it return the type of device that it should output for.
		Input Function – This function recognizes different inputs by the user and returns the type of input.
		Pipes Function – This function creates the pipes and allows them to interact with other characters and objects in the setting.
		Sprites Function – This function creates different sprites ranging from underwater objects to the main fish object.
		Collision Function – This function detects collisions made by the main sprite and the pipes created
		Menu Function – This function distinguishes between different menus and allows the user to pick one player or two player
		Main play Function – This function creates the game and connects the different function together. This function will be used when using black box/functional testing.

Figure 1: Software Description

3.2   Test Team

The test team will consist of Surinder Gill, Nick Lago and Joshua Hu. The three main testers will split the entire testing evenly covering all the different types of testing and

preparing the automated testing.

3.3   Milestones

The locations for testing will be centralized in our designated lab locations and work areas. Milestones and dates for the testing will be based off the different tests that will be conducted:

Table 2: Milestones

Module Testing	Expected Date of Completion
Sound Output testing	October 30, 2015
Frame resizing testing	October 30, 2015
Input data testing	November 6, 2015
Screen Output (Rendering) testing	November 6, 2015
Sequencing order testing	November 13, 2015
Sprite Rendering testing	November 13, 2015
Cookie storing testing	November 20, 2015
Restart and loop testing	November 20, 2015

3.4   Testing: Primary

The software that will be required is a JavaScript automated testing framework, a server that can host our website when we're testing on different devices and browsers, a text editor that can make changes to specific pieces of code that will require changing, and a media editor that will be able to modify images and change signs. The required personnel to set-up automated testing will be required to have sufficient knowledge in JavaScript and HTML to setup multiple test cases. However, users and development groups that will be doing manual testing do not not have specific requirements needed for testing. The automated testing will be done through Jasmine Framework that provides its own documentation. It creates cases that require inputs and expected outputs and will allow the tester to find faults if any. It will create webpages with documentation that show whether a test case has passed or failed and why. The proof of concept will indicate correct sound output and frame resizing. It will display a simple test run through the Jasmine Framework showing that both functions are ideal in their outputs. This testing will then be shown and used as a guideline to other tests.

3.5   Testing: Secondary

Secondary testing will be done by users in our open beta testing. We will allow users to access our game at an early development stage after preliminary tests by developers. They will be able to manually perform functional testing by checking for bugs and running through the game through our website.

4 Specifications and Evaluation

4.1 Specifications

Business Functions

- The executable HTML file will create a new browser window.

- The HTML will be executed by a browser with JavaScript functionality.

- The game will have a standby state in which it waits for user input.

- Upon the reception of user input from the standby state the game will begin.

- At the beginning of the game the user will perceive all stats reset to their default state.

- At the beginning of the game the user character will maintain its state until user input is received.

- If there is a collision with the user character and an obstacle object the game will terminate and all stats will be recorded.

- Upon termination of the game state all stats will be reset to their default state and the standby state will be reinitiated.

- If there is a collision with the user character and an objective object the user's score will increment and the objective object's instance will terminate.

- During the game state reception of user input will cause the user character to respond in a constant and uniform manner relative to the user character's instance.

Structural Functions

- Cookie

    Test how cookies are created and used by the program through static and dynamic testing. This includes code analysis, unit testing, and system testing.

- Rendering

    Test the use of graphic files compared to drawing objects for the game rendering. This includes manual system testing for aesthetic purposes.

Test/Function Relationships

Much of the testing done for the structural functions will be changed depending on how we (the developers) choose to optimize or design the appearance of the product.

For the structural testing of the cookies, it must be decided on which data to store for each cookies instance and how that data will affect the game. This can be done initially through manual code analysis then verified with unit testing, and a manual system test where the value of the cookie can be checked by access through the game and of where the cookies are stored.

The structural rendering tests will be used to decide through the aesthetic decisions of the designers and developers the presentation of core functional and ornamental objects in the game. As such the tests conducted will mostly be done through manual system tests although additional unit tests can be performed to validate the functionality of either drawing or using images to render objects.

Test Progression

The tests will proceed by verifying the business functions and all critical components of the project software. With the basis of the project verified structural testing can proceed to optimize the performance of the product and the end users' experience with the product.

4.2   Methods and Constraints

Methodology

The A Team's (the developers) approach to testing is to validate the core functionality of the product being developed before testing non-critical components of the product.

Test Tools

For unit testing we will be using the testing framework Jasmine. For system testing we will be running the game by executing it on a browser and observing the functionalities. For code analysis we will use humans with coding experience to analyze the code.

Extent

The entirety of the product will be tested.

Data Recording

The results of the unit testing will be written to a test results log. System testing will be added to a separate test log specifcaly for system testing, as will code analysis.

All test will contain information on the aspect being tested, the date of the test, the person running the test, the results of the test, a description of the test purpose, a description of the test results, and next steps from the test.

Constraints

Due to the game's simple mechanics, there are few constraints on the testing.

4.3   Evaluation

Criteria

Our tests will cover primarily the boundaries of the game mechanics and some testing inside the boundaries as an example test of normal behaviour.

Data Reduction

All test logs will indicate a pass or a need fro review which allow us to focus on reimplementing and testing components which do not pass the tests.

5 Test Descriptions

5.1 Test Identification

Control: The team are going to be using automatic insertions, in the form of a unit tester named Jasmine. Our team will also manually create non automated inputs in the form of playing the game.

The inputs created from Jasmine will be in the three different forms. First will we have Jasmine change the location of our fish player to random locations, and by knowing what spaces are occupied by pipes, the Jasmine Framework can determine when the collision function should return true. Next our inputs will be in the form of Jasmine checking for stored cookies. As a result of some browsers not allowing cookies, this isn't a full proof test and some manual testing will be needed. Lastly, the input going into the input recognition function will be a replica of random inputs a user may put if they were actually playing. If the program registers the input as intended, the program will pass. For the manual testing, our inputs will be the team actually running the game and giving usual and unusual user inputs and seeing how the program reacts.

Manual testing will occur on different browsers such as Chrome, Explorer, Firefox, etc. This will allow diversity within our tests. Automated testing (through Jasmine) will consist of many different sets of inputs. Jasmine will be given a set of expected outputs and will allow a test to pass if these outputs are met given many different inputs. Some tests will attempt to generate exceptions by accessing inputs outside of the alphabet of possible inputs such as accessing cookies that do not exist, etc.

The outputs will be pretty straight forward to tell for the Jasmine Framework. If the fish is on the same location that is occupied by a pipe the collision function should return true. The output for the cookie testing will be slightly different, as it will just be a check to see if the cookies were stored. The output for our manual testing will be comprised of images showing the expected results.

5.2 Additional Test Identification

The last of our testing will be semantics and syntax testing. We need for this program to render neatly and look clean. This will be done by trying different frame sizes, different animations and changing the images in our sprite library. Checking this fairly regularly is a smart idea to find the most ascetically pleasing set of sprites and the nicest size to fit our game onto (a preferred size). Variable naming and implementation layouts will be tested to make sure our code is understandable to all members and that the code makes sense to someone trying to maintain it. Syntax will be tested regularly by running the program and having other members look over new additions. This will make sure that we avoid a

syntax mistake that could cause problems later down the road for our project and have the team unsure about its location. System tests will be conducted to cross check against the prototype to ensure proper functionality.

In addition structural testing will include performance, difficulty, accessibility, and meeting aesthetic requirements. Performance related testing will determine frame rate speeds on differing browsers; server lag and the machine's processing power will be taken into consideration. Difficulty will be adjusted for a subjectively optimal user experience by mutation testing the rendered obstacle functions. The accessibility will be tested by running system tests on different machine types running varying browsers and operating systems per test. Aesthetic requirements will be met through manual system testing by using different sprite designs and will be decided at the discretion of the developers.