SE 3XA3: Test Plan
Pong Invaders

Team 10, Ben 10
Rehan Theiveehathasan, theivers
Puru Jetly, jetlyp
Karnvir Bining, biningk

December 8, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Oct 31, 2016	1.0	Initial Draft
Dec 8th, 2016	2.0	Final Revision

1 General Information

1.1 Purpose

The purpose of testing Pong Invaders is, at its core, to ensure that the game meets all its requirements whether technical, functional, or usablilityrelated.

1.2 Scope

The scope of the project is a two phase project, that our team believes to be moderately complex. Phase one is Space Invaders, while phase two in Pong. Once both are completed the games can be merged into Pong Invaders. With our workflow being split into two phases, our team can test functionality in parallel. This also allows our team to ensure working subsystems of Space Invaders, and Pong prior to Pong Invaders being created. Once testing is complete for the subsystems, all that would remain to be tested - once Pong Invaders is created - are merge conflicts which would be trivial to test for by unit testing.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
A.I	Artificial Intelligence, is the programmed behaviour of non-player controlled entities in the game.
Eng	Engineering
ETC	Et cetera
Ex	Example
git	Git Lab
J	Java
OS	Operating System
Repo	Repository
txt	Text File
Repo	Repository

Table 3: **Table of Definitions**

Term	**Definition**
Ben Ten	Project Team
Client	The group that this product is made for
Customer/User	Any individual or group that utilizes the product after completion
Product	The game that is being developed
Project	Development of the product

1.4 Overview of Document

The purpose of this document is to describe the overall test plan on how testing for Pong Invaders is to be conducted. The following test plan described in this document will provide what framework will be used to test Pong Invaders, as well as the test cases that will be examined.

2 Plan

2.1 Software Description

Pong Invaders is written in Java, and is a game that takes mouse or keyboard as inputs. For output the game displays the game state and graphics to the display.The keyboard input will be used to move the ship in the game, navigate through menus, and fire bullets. The mouse input will be used as well if the player so desires, as the controls in game can be rebounded to the mouse. The functions of the software that are to be tested are the: window, collision, keyinput, menu,player, bullet and alien functions. "The window function" handles the window creation when the game is started. The collision function" handles all detection of sprites with one another. "The keyinput function" handles whether a key has been pressed and what to do when that key is pressed. "The Menu function" handles the main menu game state. "The player function" handles the ship functionality and how it moves. "The bullet function" handles how the ship fires bullets and how they are drawn. Lastly "The alien function" handles how the aliens are drawn on the screen and their A.I.

2.2 Test Team

The test team for Pong Invaders will consist of the initial developers: Puru Jetly, Rehan Theiveehathasan, and Karnvir Bining.

2.3 Automated Testing Approach

The automated testing approach that our team is adopting is JUnit. We will be using JUnit due to how widely used of a framework it is, as well as how easy it is to learnand implement. We plan on using JUnit to automate testing for the collision function, the alien function, and the keyinput function to see if those functions output the correct type and expected values to other functions.

2.4 Testing Tools

For testing tools we will be using JUnit

2.5 Testing Schedule

Testing will be done one week prior to Rev0 being completed.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Area of Testing1

Menu and Peripherals Test

1. Mouse Coordinate Test

   Type: Functional

   Initial State: Active Game State

   Input: Mouse Movement

   Output: Real-Time Mouse Coordinates

How test will be performed: With the game in an active running state, the program will be set to print the mouse coordinates whenever the mouse is moved. The mouse will be moved to the boundaries of the screen and it will be checked that the coordinates are equal to the screen length and width of the screen when respectively checked.

2. Input Key Test

   Type: Functional

   Initial State: Active Game State

   Input: Keyboard inputs

   Output: Playable character Movement

   How test will be performed: The game will be put in an active state and it will be checked that all pre-set keys are functioning and the correct output is givenin game. For example if the left and right arrow keys are the key bindings for ship movement then the ship should move on screen when keyboard input is given.

3. Menu Test

   Type: Functional

   Initial State: Active Game State

   Input: Keyboard and mouse inputs

   Output: Correct Menu option is followed

   How test will be performed: The game will be put in an active menu state and the menuscreeen will be used to traverse different menu options. The Play now option will be checked to start the game. Followed by the how to play option which should display a how to play guide. Finally a re-binding option should be given so that the player can re-bind input keys for personal preference during play.

3.1.2 Area of Testing2

Ingame Value Test Cases

1. Method Check

   Type: Functional

   Initial State: In-Active Game State

   Input: JUnit test cases

   Output: Relevant return values for tested methods

   How test will be performed: The game will be left in an inactive state. Several J-unit tests will be set up to give major methods values and check for the correct return values.

2. Collision Tests

   Type: Dynamic

   Initial State: Active Game State

   Input: none

   Output: Collision identification dialogue box or print statement

   How test will be performed: The game will be in an active state and the player wil be set to simply shoot bullets and stay in a static position. The program will be set to print each time a successful collision is made. After 3 successful bullet collisions, the program will stop bullet shots and wait for the enemies to hit the playable ship to check for final collision.

3. Exit Game Test

   Type: Functional

   Initial State: Active Game State

   Input: Keyboard and mouse input

   Output: Game exits

   How test will be performed: After entering a running game, the user will exit using the associated key binding or by exiting the window entirely. In the first case, the user should return to the title screen with saved settings intact. In the second case, the user should be able to turn the game back on with saved settings intact.

4. Game Over Test

   Type: Manual

   Initial State: Active Game State

   Input: none

   Output: Game Over Screen

   How test will be performed: Game start will be initiated and aliens will simply run down the screen until user character is hit. The collision should be followed shortly by a game over screen.

5. Invalid Key Test

   Type: Functional

   Initial State: Active Game State

   Input: Keyboard input

   Output: Error messages

   How test will be performed: Invalid keyboard inputs, unbound keys, will be input while game is in progress. If an invalid key is pressed twice or more a small dialogue box will appear remind the user that the key is unbound.

6. Sprite Test

   Type: Functional

   Initial State: Inactive Game State

   Input: none

   Output: Correct object sprite

   How test will be performed: The program will be set to display each sprite when called from a j-unit test. Each sprite should match the true case in the j-unit test.

3.2 Tests for Nonfunctional Requirements

3.2.1 Area of Testing1

Stylistic tests

1. Input Test

   Type: Manual

   Initial State: Active Game State

   Input: Mouse Movements and keyboard input

   Output: real-time input tracking

   How test will be performed: multiple user inputs will be given, the system will have to print back all given inputs in real-time.

3.2.2 Area of Testing2

Maintainability and support tests

1. Legal Test

   Type: Manual

   Initial State: Active Game State

   Input: Mouse Movements and keyboard input

   Output: none

   How test will be performed: Game will be checked for any legal issues, no offensive language or icons

2. Development Test

   Type: Manual

   Initial State: Inactive game state

   Input: none

   Output: none

   How test will be performed: The game jar will simply be downloaded on a none master branch computer and checked to see if local development is possible. To allow for user development and personal use.

4 Tests for Proof of Concept

4.1 Area of Testing1

Space Invaders Testing

1. Movement of ship

   Type: Functional

   Initial State: Application is running

   Input/Condition: Keyboard buttons a and d'

   Output/Result: Ship moves

   How test will be performed: Test will be done manually by developers running prototype.

2. Hitbox of alien

   Type: Functional

   Initial State: Application is running

   Input/Condition: Keyboard spacebar

   Output/Result: Ship shoots an alien. Alien dies.

   How test will be performed: Test will be done manually by developers running prototype. Alien will be lined up with ship. Ship will then shoot a bullet towards to alien to check for correctness of hitboxes on alien.

3. Hitbox of ship and end game screen

   Type: Functional

   Initial State: Application is running

   Input/Condition: Alien is on course to hit ship.

   Output/Result: Ship takes damage. Player loses game

   How test will be performed: Test will be done manually by developers running prototype. Ship will not move when game starts and Alien will hit the ship. Test done to see if end game screen can be reached by destroying the ship.

4.2 Area of Testing2

Pong Testing

1. Hitbox of paddle

   Type: Functional

   Initial State: Application is running

   Input/Condition: Pong paddle moves with w and s and can hit pong ball

   Output/Result: Pong paddle moves up and down on left side of screen and hits ball back

   How test will be performed: Test will be done manually by developers running prototype. Ball is set on course to hit paddle to check if physics on paddle are working.

2. Hitbox of the alien

   Type: Functional

   Initial State: Application is running

   Input/Condition: Pong paddle hits pong ball towards an alien.

   Output/Result: Alien blows up.

   How test will be performed: Test will be done manually by developers running prototype. Ball is set on course to hit paddle then it rebounds to an alien. This test done to check if hit boxes on alien interact with pong ball.

5 Comparison to Existing Implementation

The implementation of our game is somewhat different than the original. However, all requirements of the original have stayed the same in this implementation. One difference from the existing implementation is that there is a Pong overlay. Now the user must destroy the aliens coming for the ship while also playing a game of Pong at the same time. The Pong ball can interact with the aliens, which means that the ball can destroy aliens while also bouncing back to the original spot it came from. Another difference from the existing implementation is that the resolution of the game has increased from 800x400 to 1280x720. This way graphics can be presented in higher definition. Also the smoothness of the game has been increased by increasing the frames per second from 30 to 60.

6 Unit Testing Plan

6.1 Unit testing of internal functions

1. Case 1

   Type: Functional

   Initial State: Ships coordinate in x and y selected.

   Input/Condition: Ship object.

   Output/Result: Checks to see if Ships in x and y coordinates are drawn correctly to window.

   How test will be performed: Ships coordinates in x and y will be selected before start of program.

2. Case 2

   Type: Functional

   Initial State: Aliens coordinate in x and y selected.

   Input/Condition: Alien object.

   Output/Result: Checks to see if aliens in x and y coordinates are drawn and are equal distant apart and drawn to window.

   How test will be performed: Ships coordinates in x and y will be selected before start of program.

3. Case 3

Type: Functional

Initial State: Pong Paddles and Pong ball coordinate in x and y selected.

Input/Condition: Paddle and ball object.

Output/Result: Checks to see if paddles in x and y coordinates are drawn and are equal distant apart, also checking if ball spawns at correct spot and drawn to window.

How test will be performed: Paddles and ball coordinates in x and y will be selected before start of program.

4. Case 4

   Type: Dynamic

   Initial State: Bullet shot from ship.

   Input/Condition: User shoots bullet.

   Output/Result: Bullet does not change its x coordinate, y coordinate increases and then bullet is out of view. Bullet is then undrawn from to save computation.

   How test will be performed: The user will shoot a bullet from ship

5. Case 5

   Type: Dynamic

   Initial State: Application is started.

   Input/Condition: No input.

   Output/Result: Fps counter on left side of screen shows if program running at 60fps.

   How test will be performed: Application just needs to be run.

6. Case 6

   Type: Dynamic

   Initial State: Ship is drawn to correct x and y position.

Input/Condition: User tries to move ship with keys a and d and shoots bullets with spacebar

Output/Result: Ship moves to desired position of left or right. Ship shoots bullets when desired by user.

How test will be performed: The user will move ship with keyboard and shoot with keyboard.

7. Case 7

   Type: Dynamic

   Initial State: User pong paddle is drawn to correct x and y position.

   Input/Condition: User tries to move paddle with keys w and s

   Output/Result: Users paddle moves up and down when desired by user.

   How test will be performed: The user will move paddle with keyboard.

8. Case 8

   Type: Functional

   Initial State: No game objects in handler class.

   Input/Condition: Program attempts to add a game object in linked list.

   Output/Result: Game Object is added correctly to linked list.

   How test will be performed: Elements in linked list printed to console.

9. Case 9

   Type: Functional

   Initial State: 2 game objects in handler class.

   Input/Condition: Program attempts to delete a game object in linked list.

   Output/Result: Game Object is deleted correctly from linked list.

   How test will be performed: Elements in linked list printed to console.

6.2 Unit testing of output files

N/A

7 Appendix

This is where you can place additional information.

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.

7.2 Usability Survey Questions?

This is a section that would be appropriate for some teams.