SE 3XA3: Development Plan
Pong Invaders

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

Contents

List of Figures

List of Tables

1 Revision History

Table 1: Document Revision History

Date	Revision	Author
December 8th, 2016	1	Puru Jetly
December 8th, 2016	1	Karnvir Bining
December 8th, 2016	1	Rehan Theiveehathasan

2 General Information

2.1 Summary

The Pong Invaders project was designed to be a twist on two classic arcade games. Ben 10 as group merged together Pong and Space Invaders to create Pong Invaders; which has the user playing both of the iconic games at the same time. The game ends when the user either destroys all the aliens and defeats the AI paddle, leading to a victory. Or if the user loses to the AI or the aliens make a landing the user loses. Pong Invaders was tested using the following methods: functional testing, unit testing, system tests, error catching and manual testing. Many of the problems were found with functional tests throughout the the project and were fixed along the way. Pong Invaders came out to be a strong representation of what we at Ben 10 set out to accomplish and the test report reflects that.

2.2 Definitions, Acronyms, Abbreviations, Symbols

Please refer to Table 2 for the list of symbols, abbreviations and any terms that need defining that are used throughout the remainder of the document.

Table 2: List of Symbols, Acronyms and Abbreviations

Symbol	**Description**
Pong Invaders	The project's name
Ben 10	The group working on and developing the project
Functional Test	An input/output blackbox type test
Structural Test	A code-based whitebox type test
Unit Test	A small series of tests for functions and methods: done with Junit
Dynamic Test	Requires code/program to be executed
Static Test	Code inspection: no execution
Manual Test	Hand-written test cases
System Test	Blackbox Tests testing the finished product as a whole
Stress Test	Testing the limits

2.3 Environment

Pong Invaders was made to be used on any programming environment given that it has access to the java JVM.

2.4 References

Pong Invaders was based off of an open source project: spaceinvaders-101-java by Marc Liberatore. The license allows us to use and modify the spaceinvaders-101-java so long as the original authors and contributors are mentioned.

3 Analysis Summary

3.1 Capabilities

Pong Invaders again, is capabable of running on mac, linux or pc as long as it has access to the java JVM. The user simply needs to run the executable jar file and begin playing to their leisure.

3.2 Deficiencies

Unfortunately due to time constraints, Ben 10 was unable to provide the full experience we set out to accomplish. Pong Invaders only includes the destroy all enemies play mode and does not implement a scoring system with a "horde mode" that was also originally outlined. Furthermore, music and enemy collision visuals were kept in the waiting room and pushed back until they are able to be implemented.

3.3 Risks

Pong Invaders holds some risks in its scalability as it runs using a linked list. This may prove to be a problem when implementing a "horde mode" in the future implementation of the game. Also, the game is kept at a resolution of 1280x720 so this may prove to be a problem based on user screen size.

3.4 Recommendations and Estimates

Increasing game functionality and aestetics can be a tall task. Adding a second game mode can take upwards of 4-5 hours as changes may need to be made to the core of the game. Adding additional visual effects such as; enemy object explosions, music,

and menu visuals, is not as intensive. These changes are estimated close to 2-3 hours of time and can be added in the near future.

3.5 Option

Pong Invaders currently requires some fine tuning to its aestetic appeal but from a functional stand point the project is a succes. With some additions and a couple more hours of work put into it, Pong Invaders will hopefully be ready for the market.

4 Testing

4.1 Black Box/Functional Testing, Table 3

4.1.1 File Input

4.1.2 System

This subsection is a summary of tests regarding the system as a whole using a black-box (functional) approach. The Test Cases refer to the number (by section/subsection) in the Test Plan.

4.2 White Box (Structural) Testing, Table 4

4.2.1 Front-End: Game Display

4.2.2 Junit Testing, Table 5

Unit testing done using Junit to test method outputs.

Table 3: File Input

Initial State: Test Types: Test Factors	Start.	Jar is executed System, functional, Dynamic Correctness, Compatibility	
Test Case #	**Input**	**Expected Result**	**Result**
3.1.1.1	mouse coordinates	Successfully found mouse coordinates, pass	
3.1.1.2	key input	Correctly recognized user input keys, pass	
3.1.1.3	mouse inputs	Successfully traversed the game menu, pass	
3.1.2.2	Key input	Successfully destroyed enemy, pass	
3.1.2.3	key and mouse input	Successfully ended game with esc key and exit button from menu, pass	
3.1.2.4	N/A	Game successfully lost, pass	
3.1.2.6	N/A	Correct sprites successfully loaded in, pass	
3.2.1.1	key inputs	inputs were successfully printed, pass	
3.2.2.1	N/A	no illegal material, pass	
3.2.2.2	N/A	Game was downloaded to non-developer pc and it worked, pass	

Table 4: Visualization of AST

Intial state: Input:	Game Active State Key inputs and mouse inputs	
Test case	**Expected result**	**Result**
Ship Movement	Horizontal movement with A, D input respectively	Passed
Hitbox of Alien	Alien box rendered correctly	Passed
Hitbox of Ship and End game	Game ends when hitbox reached	Passed
Hitbox of paddle	Ball returned correctly	Passed
Hitbox of alien	Does not pass paddle	Passed

Table 5: Parse & Lexer Structural Tests

Initial State:	N/A	Active Game State	
Test Types:		Unit, automated	
Test Factors		Correctness	
Test Case #	**Input**	**Expected Result**	**Result**
3.1.2.5	key inputs	No output when keys pressed	pass
3.1.2.1	Mouse movement	True when point is passed by mouse	pass