SE 3XA3: Test Plan
Title of Project

Team #, Team Name
Student 1 name and macid
Student 2 name and macid
Student 3 name and macid

December 8, 2016

Contents

1 General Information 1
1.1 Purpose . . . 1
1.2 Scope . . . 1
1.3 Acronyms, Abbreviations, and Symbols . . . 1
1.4 Overview of Document . . . 1

2 Plan 2
2.1 Software Description . . . 2
2.2 Test Team . . . 2
2.3 Automated Testing Approach . . . 2
2.4 Testing Tools . . . 3
2.5 Testing Schedule . . . 3

3 System Test Description 3
3.1 Tests for Functional Requirements . . . 3
3.1.1 User Input . . . 3
3.2 Tests for Nonfunctional Requirements . . . 6
3.2.1 Usability . . . 6
3.2.2 Performance . . . 6

4 Tests for Proof of Concept 7
4.1 Game Logic and Controller . . . 7

5 Unit Testing Plan 8
5.1 Unit testing of internal functions . . . 9
5.2 Unit testing of output files . . . 9

6 Appendix 10
6.1 Symbolic Parameters . . . 10
6.2 Usability Survey Questions? . . . 10

List of Tables

1 **Revision History** . . . ii
2 **Table of Definitions, Abbreviations, and Symbols** . . . 2
3 **Testing Schedule** . . . 3

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
10/21/2016	Daniel Agostinho	Initial Draft
10/21/2016	Anthony Chang	Initial Draft
10/21/2016	Divya Sridhar	Initial Draft
11/13/2016	Daniel Agostinho	Updated Gantt
12/05/2016	Divya Sridhar	Updated Test Plan

1 General Information

1.1 Purpose

The purpose of this document is to outline the testing, validation, and verification procedures that were implemented throughout the course of our project, PROJECT TETRIS. The means of testing outlined in this report were used to ensure that the game play process is as smooth as possible, and to ensure that implementation of the game was done successfully. Outlined at the beginning of this document is a list of changes made to this section of project, as this is a revised document based on the final code of PROJECT TETRIS.

1.2 Scope

The Test Plan is a basis for testing the functionality and successful re-implementation of the classic game Tetris, called PROJECT TETRIS in this case. It has the primary objective of proving that PROJECT TETRIS has met the requirements that have been specified in the requirements document. It must also adhere to the metrics applied to those requirements, so that our success is quantifiable and can be measured.

The testing plan acts as a means to arrange testing activities. This document presents the main aspects of our game that are to be tested. It is also an outline that will be followed of the different testing methods and tools used throughout.

1.3 Acronyms, Abbreviations, and Symbols

See Table 2.

1.4 Overview of Document

PROJECT TETRIS is a re-implementation of the classic game Tetris based on an open source code. The software will allow users to play the game on any machine that runs Java. As Tetramino falls and sets on the well, the user is able to change its direction and orientation using the arrow keys. The software's full requirements can be found it the Requirements Document.

Table 2: **Table of Definitions, Abbreviations, and Symbols**

Abbreviation	**Definition**
PoC	Proof of Concept
SRS	Software Requirements Specification
GUI	Graphical User Interface
Functional Test	Input/output black-box test
Unit Test	Small string of tests that examines functions and methods
Stress Test	Testing the limits of the program; this is often done using a large length of time or data
Dynamic Test	Testing that requires the program to be executed
Static Test	Inspection of the code without execution
System Test	Black-box testing of the product examining the products functionality as a whole rather than its internal components
Automated Test	Testing done by using a framework; this project will be utilising JUnit

2 Plan

2.1 Software Description

This software is a game that allows the user to play the classic game Tetris. The implementation will be completed in Java, thereby allowing this version of the game to be run offline and on any platform containing a JVM.

2.2 Test Team

The individuals responsible for testing are Divya Sridhar, Daniel Agostinho, and Anthony Chang.

2.3 Automated Testing Approach

The primary automated testing tool used was JUnit. Test classes were created for central game logic functions, such as horizontal translation by one unit and vertical translation by one unit. Other functions that required testing include

2.4 Testing Tools

Elaborate - CM
The tool used for this project is JUnit; it will be utilized to automate the unit testing.

2.5 Testing Schedule

Table 3: **Testing Schedule**

Task	**Team Member**	**Date**
User Input	Anthony Chang, Divya Sridhar, Daniel Agostinho	10/31/2016
Game Output	Daniel Agostinho, Anthony Chang	11/04/2016
Usability	Divya Sridhar	11/08/2016
Performance	Divya Sridhar, Daniel Agostinho, Anthony Chang	11/11/2016

A more detailed Gantt chart can be found here.

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

User Input and Game Output

1. KI-1
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: None

   Output: Tetraminos continue progressing downward until they hit bottom of well

   How test will be performed: The function that waits for a keyboard input in order to modify either the configuration, speed or location of the piece will receive no input, and the view of the game will reflect that the piece stays in the same horizontal position and configuration.

2. KI-2
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Right arrow pressed

   Output: Tetramino on screen moves one space to the right

   How test will be performed: The function that waits for a keyboard input in order to modify the location of the piece will receive an input, and the view of the game will reflect that the piece has moved one space to the right.

3. KI-3
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Left arrow pressed

   Output: Tetramino on screen moves one space to the left

   How test will be performed: The function that waits for a keyboard input in order to modify the location of the piece will receive an input, and the view of the game will reflect that the piece has moved one space to the left.

4. KI-4
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Spacebar pressed

   Output: Tetramino on screen accelerates downward at a faster rate than 1 space per second

   How test will be performed: The function that waits for a keyboard input in order to modify the location of the piece will receive an input, and the view of the game will reflect that the piece is accelerating downward.

5. KI-5
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Up arrow pressed

   Output: Tetramino on screen rotates clockwise once

   How test will be performed: The function that waits for a keyboard input in order to modify the configuration of the piece will receive an input, and the view of the game will reflect that the piece has rotated once clockwise.

6. KI-6
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Down arrow pressed

   Output: Tetramino on screen rotates counter-clockwise once

   How test will be performed: The function that waits for a keyboard input in order to modify the configuration of the piece will receive an input, and the view of the game will reflect that the piece has rotated once counter-clockwise.

7. KI-7
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Any other key is pressed

   Output: Tetraminoes continue progressing downward until they hit bottom of well

   How test will be performed: The function that waits for a keyboard input in order to modify either the configuration, speed or location of the piece will receive an input that is not valid, and the view of the game will reflect no change in the original horizontal location and configuration of the Tetramino.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability

Test for Usability

1. U-1
   Type: Structural, Static, Manual

   Initial State: The game has not been compiled or run and the user has the program on their computer.

   Input/Condition: compile and run the program on Mac OS X, Windows, or Linux

   Output/Result: The program should compile and launch PROJECT TETRIS

   How test will be performed: The program will be compiled and launched on each different platform to ensure the usability of the game by anyone who wishes to use it.

3.2.2 Performance

Test for Performance

1. P-1
   Type: Structural, Static, Manual

   Initial State: The game testers will have no previous knowledge of how PROJECT TETRIS works.

   Input/Condition: Users will play PROJECT TETRIS for 2 minutes
   Output/Result: The amount of rows cleared in 2 minutes is a minimum of 4

   How test will be performed: The program will be tested by people who have never played PROJECT TETRIS. If they are able to play such that they are able to clear at least 4 rows in under 2 minutes, it proves that the game is easily enough to understand without prior explanation or use of instructions.

4 Tests for Proof of Concept

The Proof of Concept testing is focused mainly on the game logic working and getting the keyboard input to work. There is no view aspect to the game and the only way to verify that the program is functioning is through the console. The tests used for the Proof of Concept are similar to the tests for the Key Input tests.

4.1 Game Logic and Controller

1. POC-1
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: None

   Output: The console will output that the Tetramino is moving downward.

   How test will be performed: The function that waits for a keyboard input in order to modify either the configuration, speed or location of the piece will receive no input, and the console will display that the Tetramino is only moving downward.

2. POC-2
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Right arrow pressed

   Output: Console outputs that the Tetramino moves one space to the right.

   How test will be performed: The function that waits for a keyboard input in order to modify the location of the piece will receive an input, and the console will display that the Tetramino has moved one space to the right.

3. POC-3
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

Input: Left arrow pressed

Output: Tetramino on screen moves one space to the left

How test will be performed: The function that waits for a keyboard input in order to modify the location of the piece will receive an input, and the console will display that the Tetramino has moved one space to the left.

4. POC-4
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Up arrow pressed

   Output: Tetramino on screen rotates clockwise once

   How test will be performed: The function that waits for a keyboard input in order to modify the configuration of the piece will receive an input, and the console will display that the Tetramino has rotated once clockwise.

5. POC-5
   Type: Functional, Dynamic, Manual

   Initial State: No keyboard input given

   Input: Down arrow pressed

   Output: Tetramino on screen rotates counter-clockwise once

   How test will be performed: The function that waits for a keyboard input in order to modify the configuration of the piece will receive an input, and the console will display that the Tetramino has rotated once counter-clockwise.

5 Unit Testing Plan

For unit testing of this project, Team TETRIS will use JUnit.

5.1 Unit testing of internal functions

Certain functions within our program receive input and return output. There is a certain behaviour that we wish those functions to follow. For a given input there, there is a corresponding expected output. We can unit test the internal functions by giving these functions inputs that will throws exceptions (as expected) and inputs that will yield expected results. No stubs or drivers will be needed for testing. Our goal is to cover at least 75% of our code to ensure that most types of functions are tested.

5.2 Unit testing of output files

Since the output of the game is a view of the current state of the game, it is an integration of all the parts of the program working together. The only way to unit test the output is to give the internal functions an input and compare the output to how we expect the output to behave. For example if we give as input Tetramino move right, we can test that it works by checking if the output has the Tetramino moving one space to the right. Another example is if the Tetramino is at the right edge of the screen and we give it the input to move right again. We can verify that the program works if the output shows that the Tetramino does not move right and conforms to the physical laws of the game.

5.3 Usability Survey Questions?

Survey isn't styled, is ambiguous, open-ended questions may not give good results, not linked to NF tests - CM

Have you played the original Tetris game? How does this version compare to the original? Does it look and feel like the original Tetris? Was it difficult to learn how to play and understand the rules?