DEFENDER FINAL BATTLE: Project Test Report

Sheena Philip (545819) Moolisa Tlali (564988)
School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg
ELEN 3009 : SOFTWARE DEVELOPMENT II

1 Introduction

Testing of the developed source code is conducted to examine the performance reliability of the code and to increase confidence in what it is set out to do. Automated unit test suite has been developed to ensure that a single unit works correctly and as expected. The test suit automatically runs through test pertaining to each unit and makes available the results of that unit. It also provides results for all the tests conducted. In this object oriented solution a unit is defined to be a class. Thus the confidence in the developed solution cannot be set-out until all classes that form the basis of game functionality and those that determine test the boundary conditions of the solution perform as required. This report therefore details an overview on the testing process conducted and an evaluation of this testing process.

2 Testing Environment

The Automated unit testing employed in the solution utilises Google C++ Unit Testing Framework (Version 1.7) also known as googletest [1]. It offers advanced testing functionality but for this solution only simple testing will be performed. The testing framework is used together with the CodeLite V8.0 IDE. Unit tests are processed on a system running the 64 bit version of a Windows 8 operating system. In order to run the test code, it is mandatory that the respective libraries be installed and the necessary classes be added to the project. The unit test suite provided is for the final version (defender reloaded) of the game.

3 Testing Overview

A total of 43 tests have been conducted for the six individual units/classes. Table 1 below gives a summery of the classes and the number of tests performed for each class.

TABLE 1
Divison of Unit tests

Class/Unit	**Number of tests**
SpaceFighter	6
Lander	12
Laser	4
Missiles	8
Collision	5
Keyboard	8
Total	43

The test code has been limited to classes of entities that form the basis of the game. These classes together with the collisions class test the logic and the boundaries of the game. Keyboard tests test the success of interaction between the player and the game. A basic overview of tests conducted on each class is presented below.

3.1 SpaceFighter

The SpaceFighter functionality is essential since this is the class that the player controls. It is responsible for the spaceship movements on the screen. The test first determine whether the spaceship is correctly positioned on the screen upon its creation. The four possible movement directions (UP, DOWN, LEFT, RIGHT) of the spaceship are then tested. These movements utilise the spaceship speed to determine where it should be after a specific movement. These tests pass if the x and y position of the spaceship are correctly incremented or decremented.

3.2 Lander

The lander class is first tested for correct movements. All the eight possible movements are tested and are checked against the correct increment and or decrement in the x and y coordinates of the lander. These tests pass if the correct increments have been implemented.

The Last four tests ensure that the landers do not exit the boundaries of the screen. The logic prohibit the landers to never cross the top, bottom, left and right boundaries of the 1600 by 800 pixel screen. These tests pass when the lander's x and y position is not incremented or decremented when attempting to cross the screen boundaries.

3.3 laser

The lasers, which are the SpaceFighter ammunition, are restricted for motion only in the right and left directions. The first two tests determine whether the laser has been correctly moved against the speed of the laser in the right and left directions. These tests pass if the x and y positions have been correctly incremented or decremented. The last two test determine if the lasers can be moved in the prohibited directions; up and down. They pass when the coordinates of the lander do not change even when the lander is directed to move in these prohibited motion directions.

3.4 Missiles

These constitute the lander's ammunition. on creation, they are positioned to the position of the lander and can fired from a lander at any position on the screen in the direction of the spaceship. Eight cases are set which define the position of the lander with reference to the space ship position. These cases together with the missile speed and the gradient joining the missile and the spaceship are used to perform required decrements and increments of the missile's x and y coordinates. This the eight tests determine the correct increments of the coordinates and pass if they are correctly adjusted.

3.5 Collision

This class is based on entity edge detection. There are 5 different possibilities of collision detections that can occur. These are collisions of the four game entities which constitute collisions spaceship-lander, lander-laser, spaceship-missile and laser-missile. These possibilities need not to be tested individually since the collisions class is general to any entity as long as the struct positions, widths and heights of the entities are provided.

Therefore only the possible edge collision directions are tested, that is entity1's bottom colliding with entity2's top,entity1's left colliding with entity2's right and entity1's right colliding with entity2's left. All these cases are tested by creating objects whose respective edge collide/coincides and those whose edges do not coincides. Test pass if collision is detected for coinciding edges and pass if collision is not detected for edges that do not coincide.

3.6 Keyboard

Keyboard tests are performed as these align the interactions of the player with the game. First the player keyboard states are tested for whether they have been set on(true) or are off (false). The setting of these keys will allow proper updates of the movements of the spaceship. The tests pass if the states of the keys has been set.

As well, the initial conditions of the state of the keys are tested. These are set by the constructor of the keyboard class. These tests will ensure proper initialisation of the game. They pass if the correct state of the keys is attained upon creation og the class object.

4 Critical Analysis and Evaluation

Each of tests performed received a "RUN OK" result which indicates that the tests run as expected.

The management class which contains the logic has not been tested as it also utilises the already tested objects and logic function thus will not run these objects beyond what is required of them since the objects have been tested against such attempts. Furthermore, testing between the collaboration of the game presentation layer and logic layer has been neglected due to the fact that the objects used have already been tested. What can be tested is to test whether the combined functionality these layers works. However, time constraints did not allow for such tests to be conducted. Testing by inspection/observation is conducted where combined functionality can be judged from the screen.

The application layers have been separated to a large degree, thus the model logic layer was simple to test as it stands independently from rendering any logic.

The testing can be improved by performing graphical unit testing and using integration testing methodology

5 Conclusion

The tests conducted have been used to verifying the functionality of the methods employed on the individual classes. A total of 43 successful tests across 6 classes have been conducted on the logic layer front and initialization layer front. However, further testing is required to supplement the already established validity of the software. Combined functionality between the presentation and logic layer has not been tested but subtle tests have been conducted by observation of the game play as presented on the screen.

References

[1] Googletest, "googletest:google c++ testing framework @MISC." [Online]. Available: http://code.google.com/p/googletest/