Defender V4.0 Test Report

Catherine Honegger 566247 and Christian Kamwangala 685344

School of Electrical and Information Engineering, University of the Witwatersrand, Johannesburg

1. Introduction

This report provides an overview of the unit tests that were implemented to test the reliability of the Defender V4.0 game that was developed. The unit test suite presented in this document is used to ensure that the behaviour of the Defender V4.0 code is correct. This is especially useful if any adaptations to the code are implemented in order to ensure that no future violations of the current game occur. The major logic and data classes in Defender V4.0 are tested rigorously to verify the behaviour of the code.

The test environment is described in Section 2.. Section 3. gives an overview of the tests that were implemented as well as any tests that were not implemented. Explanations of complex tests are given in Section 4., and an evaluation of the test framework is presented.

2. Test Environment

The 64 bit Windows 7 Operating System was used during the testing implementation. The test suite presented is tested using Google's framework for writing C++ tests, googletest version 1.7 with the CodeLite IDE version 8.2.0 and compiled using MinGW version 4.7. An attempt was made to include test using googlemock version 1.6, but the tests would not compile. The final version of the Defender software, version 4.0 is tested.

In order to use the unit test suite, the linker library search path must point to where the gtest lib and gmock folders are, and the compiler include path must point to where the gtest include and gmock include folders are. The necessary libraries files from the executables folder and the source code files from the test-source-code folder must be in the project folder in order to run.

3. Testing Overview

In order to ensure that there is a comprehensive coverage of the Defender V4.0 software, unit tests were compiled for all logic and data layers of the code. All classes that deal with the user interface were not tested as the googletest framework does not cater for these, and it is easy to deduce that if there is an error on screen and all unit tests pass, the error lies in the presentation layer of the code.

Individual class behaviour is tested using googletest to ensure that the game produces the expected results. The advanced automated googlemock framework was intended to be used to test difficult-to-test functionality that involves the interaction between classes and randomness. The tests have been kept as simple as possible in order to reduce the possibility of errors in the test code. Tests were only conducted on the public functions of the classes to ensure that the code is not brittle.

3.1 Bullet Class

The initialization of the Bullet class is tested to ensure correct initialization takes place. The test is verified by passing the relevant parameters to a respective pointer object and then using getters of the class. Position, ID, object size and speed are the properties tested.

3.2 CollisionDetection Class

The CollisionDetection tests determine whether a correct collision returns a 'TRUE' statement, and whether a non-collision returns a 'FALSE' statement.

3.3 Enemy Class

The initialization of the Enemy class is tested to ensure correct initialization takes place. The test is verified by passing the relevant parameters to a respective pointer object and then using getters of the class. Position, ID, object size, current direction, speed and time elapsed since last change of direction are the properties tested.

3.4 EnemyBullet Class

The initialization of the EnemyBullet class is tested to ensure correct initialized takes place. The test is verified by passing the relevant parameters to a respective pointer object and then using getters of the class. Position, ID, object size, speed and target direction are the properties tested. The velocity of the EnemyBullet has not been verified.

3.5 Entity Class

The initialization of the Entity class is not tested, as it is a parent class. Tests were conducted to verify the behaviour of getters and setters of the class. Position, object properties (including ID and object size), rotation, current direction, time elapsed and target position are the properties tested. The isActive and whenKilled methods were also tested to ensure that the entities lose their lives once they have been killed, and still have their lives if they have not been killed.

3.6 HeatSeek Class

The initialization of the HeatSeek class is tested to ensure correct initialized takes place. The test is verified by

passing the relevant parameters to a respective pointer object and then using getters of the class. Position, ID, object size and speed are the properties tested.

3.7 PlayerShip Class

The initialization of the PlayerShip class is tested to ensure correct initialized takes place. The test is verified by passing the relevant parameters to a respective pointer object and then using getters of the class. Position, ID, object size, speed and lives are the properties tested. The whenKilled method is tested to ensure that when the player dies they lose a life.

3.8 PowerUP Class

The initialization of the PowerUP class is tested to ensure correct initialized takes place. The test is verified by passing the relevant parameters to a respective pointer object and then using getters of the class. Position, ID, object size and speed are the properties tested.

3.9 Score Class

This class is tested to ensure that the player's score increases according to the rules. The following behaviours were tested:

- The correct score increase is given depending on the objects killed, fired or collected by the player.
- Saving of high score to a text file.
- Loading of high score from text file.

3.10 StopWatch Class

The start/stop functionality of the StopWatch class is tested to ensure that the time increases.

3.11 Classes not Tested

The following classes were not tested as they are part of the presentation layer, and their structure does not allow easy testing without inclusion of SFML. These classes are mainly used to display the game on the screen, and deal with keyboard inputs. The correct behaviour of these classes is primarily dependent on the correct working of both the logic layer, and the graphics library. SFML is considered to be quite reliable, and due to our unit tests above, the logic layer has been extensively tested.

- Event Handler Class
- GameEngine Class
- Game Presenter Class
- Game View Class
- Main Class
- Resources Holder Class
- ScreenManager Class

4. Complex Test Explanations

4.1 Controller Class

Movements of entities are calculated by the Controller class. Thus movement of the player's bullets, enemies, enemy bullets, heat seeking missiles and the player ship are tested.

The following behaviours of the Controller class were tested:

1. The movement of Enemy, PlayerShip and Bullet objects in a specific direction, (up, down, left and right), results in those objects being moved by their speed in that direction.
2. The movement of EnemyBullet objects towards a specific location results in the object being moved by their speed and calculated velocity in that direction.
3. The movement of HeatSeek objects towards a specific location results in the object being moved by their speed and calculated velocity to the correct position.
4. None of the objects go out of scope of the game boundaries.

In order to test behaviour 1, the PlayerShip entity is selected, and verification of a movement is done by testing whether the new position is the speed of the PlayerShip away from the old position. i.e if the object is going left, test that the y co-ordinate remains the same, but the x co-ordinate is decreased by the object's speed.

Behaviour 2 is tested by ensuring that the bullet is between the initial location, and the intended target location.

Similar to the tests for behaviour 2, in order to test behaviour 3, verification of a movement is done by testing whether the new position is closer to the intended position.

Behaviour 4 is tested using the PlayerShip to ensure that when the ship is told to move beyond a boundary, it remained in the same location.

4.2 Logic Class

The most important behaviours that were tested for this class are:

- The ability to create a Bullet and add it to a vector.
- The ability to create an Enemy and add it to a vector.
- The ability to create an EnemyBullet and add it to a vector.
- The ability to create a HeatSeek and add it to a vector.
- The ability to create a PowerUp and add it to a vector.
- The ability for the Enemy to Move.
- The ability for the Player to Move.
- The ability for the Bullet to Move.
- The ability for the EnemyBullet to Move.
- The ability for the HeatSeek to Move.

5. Conclusion

The application layers in the Defender V4.0 software are separated to a great extent, allowing for ease of logic testing.

Simple defence testing was implemented to ensure correct implementation of the game framework in the future. These tests included initialization tests of the entities within the game. The logic layer is tested extensively, with a close look at the movement and collision of all entities.

All of the tests in the unit testing suite pass as expected, returning the desired outputs and validating the software. In order to completely verify the behaviour of the game, further tests will be required if any changes are made to the software.