A2

UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Software Testing
Cohort 2 - Group 13

TAKEN OVER FROM GROUP 16

Group Members:

Carys Hoile
Ivo Hadley
Shravani Baviskar
Alex Kleijwegt
Owen Codrai
Caner Cetinkaya
Haiqal Mohammad Nazil

a)
Tests were black box tests which focused on testing the functional requirements, a traceability matrix was created to track which requirements were fulfilled by which test. It was aimed that each functional requirement would have at least one test and that each test linked to a functional requirement. As we only had a short amount of time to do testing, this helped to prioritise important parts of the code to test.

During testing, achieving 100% coverage of the code was not a significant priority, and in some cases the logic of a game class was removed from the class and placed inside a fake class. This helped deal with difficult dependencies or keyboard inputs whilst still testing the important parts of the code.

The process began with building small unit tests to test specific methods within classes, these included testing that assets were in the correct folder, player movement and methods within the SoundManager class. These tests were very important as they helped to identify specific bugs and ensure that it is easy to understand where these errors came from. It was aimed that these tests would take up 80% of the code as they were small and simple and tested basic methods in the class [1]. It's important that these basic methods are correct before trying to deal with the interactions between classes and so unit tests for a class need to cover all the specific features of that class [2].

The next tests built dealt with the interactions between classes. Primarily, these classes dealt with the different screens in the game and how these screens interacted with different classes.

Manual tests were used to check things that were not able to be tested automatically, such as checking that assets had rendered correctly or that the map was in a top down view. Descriptions of these tests can be found here: https://alexkleijwegt.github.io/testing.html. In this link there is also the coverage report and testing results.

When writing these tests it was important that they could be maintained and wouldn't need to be touched if the code was refactored, sometimes this wasn't feasible . Once a test was written, it was then run to check that the test worked as intended and errors were due to the system being put under test rather than the test being poorly written [1]. Tests were built with three parts: the setup, where classes needed were initialised and inputs and expected outputs were set up, the method call, where the method being tested is called with the inputs and finally the assertion part which checks that the method output matches up with the expected output [2].

For traceability, tests were given an identifier e.g. 1.1.1, The first part was the class e.g.AssetTests, the second part was the group e.g. if it tested image assets or sound assets and the 3rd part was the individual test. This helped to connect the manual and automatic tests.

Where methods had an infinite set of inputs, they were tested along the boundaries, if there was a finite set of inputs and each value in the set would have produced a different result, the tests were then tested across every possible input

b)

AssetTests - All of these were passed

- **Tests 1.1 -** These tested the presence of image assets for the map.
    - Tests 1.1.1-1.1.5 - Automatic Tests to test that image files were in the correct location
    - Test 1.1.6 - Manual Test to test that image assets have rendered correctly within the game. A manual test was chosen for this as there was no automatic way to test the rendering of the images within the game.
    - Test 1.1.7 - Manual test to check that map is from a top down view. A manual test was chosen for this as there is no way to automatically test the viewpoint of the game as it relies on what the assets look like.
    - Related Requirements: FR-NAVIGATE, FR-VIEW
- **Tests 1.2** - These tested for the presence of sound assets.
    - Tests 1.2.1-1.2.5 - Automatic tests to test that sound files were in the correct location
    - Test 1.2.6 - Manual test to test that sound loaded and played correctly within the game as there was no automatic way to test how the audio would sound within the game.
    - Related Requirements: UR-SOUND
- **Tests 1.3** - These tested for the presence of image assets for the avatar.
    - Tests 1.3.1-1.3.3 - Automatic tests to test that image assets were in the correct location
    - Test 1.3.4 - Manual test to test that avatar have been loaded correctly A manual test was chosen for this as there was no automatic way to test the rendering of the images within the game.
    - Related Requirements: FR-START
- **Tests 1.4.1** - These tested that the document to save scores was present
    - Related Requirements: FR-HIGHSCORE

SoundManagerTests - All of these were passed

- **Tests 2.1** - These tested the management of music volume
    - Tests 1.2.1-1.2.2 - Automatic tests to deal with changing music volume
    - Related Requirements:
- **Tests 2.2** - These tested the management of SFX volume
    - Tests 2.2.1 - 2.2.2 - Automatic tests to deal with changing SFX volume
    - Related Requirements: UR-SOUND

AchievementTests (split into 4 different classes)- All of these were passed

- **DistanceAchievementsTests - Tests 3.1** - These dealt with achievements relating to distance travelled
    - Test 3.1.1 - Automatic test to test setting up achievement
    - Tests 3.1.2-3.1.5 - Automatic tests to test levelling up to all possible levels
    - Related requirements: FR-STREAKS
- **EatAchievementsTests - Tests 3.2** - These dealt with achievements relating to eating
    - Test 3.2.1 - Automatic test to test setting up first time eaten achievement
    - Test 3.2.2 - Automatic test to test setting up tiered eaten achievement
    - Tests 3.2.3-3.2.6 - Automatic tests to test levelling up to all possible levels
    - Related requirements: FR-STREAKS

- **SleepAchievementsTests - Tests 3.3** - These dealt with achievements relating to sleeping
    - Test 3.3.1 - Automatic test to test setting up first time slept achievement
    - Test 3.3.2 - Automatic test to test setting up tiered sleeping achievement
    - Tests 3.3.3-3.3.6 - Automatic tests to test levelling up to all possible levels
    - Related requirements: FR-STREAKS
- **RecreationAcheivementsTests - Tests 3.4** - These dealt with achievements relating to recreational activities
    - Test 3.2.1 - Automatic test to test setting up first time doing a recreational activity achievement
    - Test 3.2.2 - Automatic test to test setting up tiered recreational activity achievement
    - Tests 3.1.3-3.1.6 - Automatic tests to test levelling up to all possible levels
    - Related requirements: FR-STREAKS

HustleGameTests - All these were passed

- **Test 4.1** - Automatic test to check that game dimensions initialised correctly
- **Test 4.2** - Manual test to check that game does not save
- Related requirements: FR-MENU2

PlayerTests - All these were passed

- **Tests 5.1** - Automatic tests to check player is initialised correctly
    - Test 5.1.1 - Checks player variables are instantiated correctly
    - Tests 5.1.2-5.1.4 - Checks player rectangles for collisions and hitboxes are set up correctly
    - Related Requirements: FR-GAME-PLAY-1, FR-GAME-PLAY-2, FR-GAME-PLAY-3, FR-GAME-PLAY-4, FR-INTERACT1
- **Tests 5.2** - Automatic tests to check player movement.
    - Tests 5.2.1-5.2.2 - Automatic tests to check player can move and can be set to not moving
    - Test 5.2.3 - Automatic test to check ability to freeze player
    - Test 5.2.4-5.2.15 - Automatic tests using a fake player class to check movement across various keyboard inputs. These tests were done in a fake class that removed the keyboard input and kept the logic. As this is a key part of the code and it is unlikely that it would change, moving the logic into another class made testing simpler, it also allowed for the code to simulate multiple keys being pressed at once with boolean values for left, right, up and down.
    - Test 5.2.16-5.2.20 - Automatic tests to check player interaction hit box
    - Related Requirements: FR-GAME-PLAY-1, FR-GAME-PLAY-2, FR-GAME-PLAY-3, FR-GAME-PLAY-4, FR-INTERACT1

GameOverScreenTests - All these were passed

- **Tests 6.1** - Automatic tests to calculate score
    - Tests 6.1.1-6.1.3 - Checks score calculation for lowest boundary value for sleep, study and relax (too little)
    - Tests 6.1.4-6.1.6 - Checks score calculation for middle boundary value for sleep, study and relax (right amount)
    - Tests 6.1.7-6.1.9 - Checks score calculation for upper boundary value forsleep, study and relax (too much)
    - Related Requirements: FR-HIGHSCORE

ScoreSystemTests - All these were passed

- **Tests 7.1** - Automatic tests for counting activity
    - Tests 7.1.1 - Checks activity counts and scores are set at 0
    - Tests 7.1.2-7.1.4 - Checks activity counts can be incremented when a event is done
    - Related Requirements: FR-HIGHSCORE

ScreenTests - All these tests were passed

- **Test 8** - Manual test to test different screens of the game
    - This was done with a manual test as it allows for the user to check that transitioning between the screens is natural and it also means that UI and UX issues on the screens can be identified and fixed and this will help improve customer satisfaction.
    - Related Requirements: FR-MENU1, FR-MENU3, FR-MENU4

EventManagerTests - All these tests were passed

- **Test 9.1** - Automatic test to ensure that events start as they should
- **Test 9.2** - Test interactions work as they should
    - This was a manual test as it makes it easily extensible, if a map is added in the future e.g. to add York City Centre or more interactable buildings are added. It also means that the map is checked for visual bugs as the player moves round the map. Finally if this is a manual test it can be used to test that no bugs occur when spamming interactions. It also ensures that the pop up text options is formatted correctly
    - Related Requirements: FR-INTERACT2, FR-SLEEP1, FR-ENERGY1, FR-ENERGY2, FR-WEEK, FR-TIME, FR-SLEEP2, FR-LEADERBOARD, FR-HIGHSCORE

The tests are all passed, where possible tests cover all boundary conditions or all options, this leads to the tests providing good coverage over a range of inputs. Tests are mostly complete, there are some methods that have not been implicitly covered, however, their functions are tested by tests which deal with other functions. There are two classes in the test folder which extend classes used in the game, these are designed to test code logic without keyboard input or certain dependencies. These may become less complete if the code is extended; however, it is easy to alter them to suit the current implementation.

References

[1] I. Sommerville, *Software Engineering*, 10th ed. Essex, England: Pearson Education Limited, 2015, pp. 226–254.

[2] T. Winters, T. Manshreck, and H. Wright, *Software Engineering at Google : Lessons Learned from Programming over Time*, 1st ed. California, USA: O'Reilly Media, Incorporated, 2020, pp. 207–310.