UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Software Testing Report

Cohort 2 - Group 17 (Rich Tea-m 17)

Group Members:

George Jopson
Ben Slater
Meg Tierney
William Potts
Jamie Burgess
Seyi Towolawi
Alex Staicu

Testing Methods and Approaches.
For our project, we decided the best way to conduct our tests was to go across each class and test each method within our class, as all the methods within the classes should fulfil the requirements of our project.

The majority of our unit tests would be done automatically within our codebase, as manually testing whether some of our classes work would take a long time and be very tedious. The automatic tests would be done for the logic side of our code, for example when interacting with a building that the correct amount of energy gets taken from the user's energy.

The remaining set of unit tests would be more visual and would have to be tested manually, such as us deciding whether the player's movement animation works and that the player would collide correctly with the buildings. A few of these manual tests were completed though white-box testing, where we would be able to see the bounds of the buildings and be able to check that the player cannot break past the boundaries of the building, however we found that with the manual tests, the tester did not really need to know much about the code itself, but more about what the screen/character/buildings should look like and this fit more into grey-box testing, which we used for the majority of our manual unit tests.

For our integration tests, we decided that they would be completed manually, as in theory if the automatic unit tests for the 2 or more classes that are being tested together were successful then being able to see this on the game screen would give us an idea on how successful the integration tests were for our game. This would be done in a more white-box style, as the tester should be able to know what needs to happen in each instance.

Our end-to-end tests would be entirely manual, where we would set a tasks for our tester to complete during a full run through of our game, focussing on either streaks, missing out an event or two, repeating one event over and over on each day and completing the game as if we were trying to complete the achievements. This would be white-box too as the testers would need to work out how to complete their different objectives per run by looking through the code.

For our testing approach we decided that we would be reactive, as due to when we were given the code, a lot of the game was already programmed so we had to write tests for software that had already been produced. The approach was to cover all of the classes within the code and fulfil all of the requirements that had been given to us by the user or the product brief.

Testing Material Project Website:
https://megant2004.github.io/ENG1-T17-Website/Part2/Assets/Deliverables/TestDesc.pdf

It is important to note that during the process we sometimes got a "Test events were not received" error when trying to run tests, but this could be avoided by simply rebuilding the tests.

Test Report
For our unit tests, we ran 71 automatic unit tests, with all of them passing. The majority of these tests were either in our Player class or our EventManager class, due to our Player class handling all of the player movement and player collision with other objects and our EventManager class handling all the different types of events and their different behaviours in regards to player energy and time. Due to the nature of our code a lot of our unit tests would be manual, as these would be on the graphical side of our project i.e making sure things load and the shown energy is the real energy. So this would make it so any of our "...Screen" classes would show as if they haven't been covered in our test report, as shown below.

Automatic Unit Tests:

Requirement	**Test Description**
FR-LEADERBOARD-DISPLAY** *Top 10 scores from previous playthroughs will be displayed.*	**LeaderboardScreenTests - testGetLeaderboard10** This tests if the top 10 saved leaderboard items are correctly loaded.
FR-HIGH-SCORE** *The system shall update the leaderboard if a new high score is achieved.*	This is tested via a combination of **SaveScreenTests - testSaveScore** which checks the high is saved and **LeaderboardScreenTests - testGetLeaderboard10** which tests if the new scores are correctly displayed in order (showing the new high score at the top)

The table above is a little snippet of our automatic tests linked with the requirements that we had written them for.
Manual Unit Tests:

Requirement	Description of Test	Passed/Failed
FR-VIEW	The game's camera should be from a 3D top down perspective.	Passed
FR-STAY-OUTSIDE	When interacting with the library, accomodation building and the piazza the user should not clip into the buildings whilst they are interacting with the building.	Passed
FR-CHOICE-POPUP	When interacting with the Computer Science building, a popup says "Study in the Computer Science building?", along with a choice of "Yes" or "No", which the user should be able to choose between.	Passed
FR-PAUSE-MENU	The user should start a new game and at 8:00 am (right at the start of the day) press the "esc" button to pause the game and whilst on the pause menu, the time shall not continue past 8:00 am	Passed
FR-START	When pressing the "New Game" button in the main menu, a how to play popup should show.	Passed
FR-NAVIGATE	The user should be able to navigate around the map with their arrow keys.	Passed
FR-DAY-END	The game should lock activities from being done at the end of the 16th hour.	Passed
FR-LEADERBOARD-DISPL	When pressing "Leaderboard" the game should show the	Passed

AY	leaderboard of all the scores.	
NFR-INSTRUCTIONS	The user should be able to read the instructions.	Passed
NFR-VISUALS	The game should present the university in a happy and positive way.	Passed
UR-MENU	The main menu has 5 buttons. “New Game”, ”Leaderboard”, ”Settings”, ”Credits” and “Exit”.	Passed
UR-CUSTOMISE	The user should pick a character and that character should appear on the map.	Passed
UR-SETTINGS	The user should be able to change their settings in the “Settings” menu.	Passed
UR-DEVICE	The game should be playable on the desktop/laptop.	Passed
UR-WORLD	The map should be an accurate 2D representation of Heslington.	Passed
N/A	Credit screen loads when the credits button is pressed.	Passed

The above table is a snippet of the manual tests that were completed fitting to the requirements of the project.
For our Integration tests, we ran some automatic tests, but the majority were manual tests. These manual tests would be making sure that when a player interacted with a building that the game screen would update all of the different changed values (time and energy), as well as popping up with the correct dialogue boxes for the correct situation. These would test the EventManager, Player, DialogueBox, GameScreen and SoundManger classes together. This would be repeated for all of the different events within our game. The only other Integration test that was needed is the saving the player’s score to the leaderboard, which would test the EventManager, DialogueBox, SoundManager, SaveScreen and LeaderboardScreen classes together. All of these tests when manually run passed.

Requirement(s)	Classes	Description	Passed/Failed
UR-TIMED UR-ENERGY UR-SOUND UR-WORLD FR-CHOICE-POPUP FR-STAY-OUTSIDE FR-NAVIGATE FR-TIME FR-SLEEP-LOCATION FR-STUDY-LOCATION FR-EATING-LOCATION FR-LEISURE-LOCATION FR-SCORE	EventManager Player DialogueBox GameScreen SoundManager	For each event* (Tree, Chest, Piazza, CompSci, RonCookeHub, Accomodation, Town, Flowers, Shop, Gym, Library, East, DuckPond), the game should update the values of the energy and time accordingly, make the correct sound and provide the correct score.	Passed
UR-MENU UR-SOUND UR-LEADERBOARD FR-CHOICE-POPUP FR-SAVED-DATA FR-WEEK	EventManager DialogueBox SoundManager SaveScreen LeaderboardScreen	When the game is finished, the user’s score should be shown and the game should allow the user to input their name into a	Passed

FR-COUNTER FR-LEADERBOARD-DISPLAY FR-HIGH-SCORE FR-SCORE FR-END-SCREEN		leaderboard. The user's name and score should be put on the leaderboard if the score is high enough.	

*Each event was its own Integration Test. So from this test we actually had 13 tests that were very similar so we decided to group them into this test.
For our E2E tests, we only ran manual tests. For these tests we set a few tasks for the player to do within a run through of the game. This was to make sure no unknown bugs would occur. These tasks were a normal run, a run only eating and nothing else, a run doing nothing but sleeping, a run only studying and a run only doing recreational activities. For the normal run, we encountered a bug on day 5, which would close the game. This was quickly fixed to ensure this test along with the others would all pass. All of these tests, after this fix, passed.

Run-Type	Description	Passed/Failed
Normal	The user should complete a normal run to try and get a high score. Expected Score: Normal	Passed**
Just Eating	The user should complete a game by only eating, then sleeping when the user has no energy. Expected Score: Low (Not 0)	Passed
Just Sleeping	The user should only sleep. Expected Score: 0	Passed
Just Studying	The user should complete a game by only studying, then sleeping when the user has no energy. Expected Score: Middle (~60%)	Passed
Just Recreational Activities	The user should complete a game by only doing recreational activities, then sleeping when the user has no energy. Expected Score: Low (Not 0)	Passed

**After the bug fix.

Code Completeness/Correctness

Our code and tests are functionally complete due to them covering all the different user and functional requirements that we had decided on. This has been due to us, focussing on each requirement and trying to link them together to a couple of other requirements to have tests that could fulfil multiple requirements at one time.
We believe that our tests are also functionally correct, as we decided on a bunch of different parts to test each method/requirement so that, when it came round to certain parameters the code didn't just break randomly on a random day/hour/activity in the game.