UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Testing

Group 15 - HesHus

Adam Robinson

Tsveta Ivanova

April Jack

Oliver Goodwin

Luke Jackson

Luca Hammond

Chris Rolfe

Our software testing for Heslington Hustle is fully traceable to every requirement identified in both versions of the game. The approach we adopted was to test both the functional and nonfunctional requirement tables that we previously updated with the additional game features and specifications our client requested. Testing is essential, to ensure maintainability and code quality and to ensure our game runs smoothly. Individual components of the game function that contain our requirements are checked thoroughly and in isolation to ensure correct behaviour and consistency.

We first employed a method to identify which requirements can be tested by unit and manual tests. Our nonfunctional requirements table contained methods suitable for manual testing. The requirements that needed manual testing related to more visual checks that required a human. For example, checking if the system is operable by users with red-green colorblindness, visually checking for screen quality, and being able to hear the sound, or checking the suitable demographic requirement promising that our game will be enjoyed by 95% of users playing it.

In addition, we performed both manual and automated testing on many features, such as game over testing and achievements, to improve coverage. The majority of our functional requirements table was suitable for automated testing. The LibGDX environment relies heavily on graphical and input/output systems, so for our automated testing, we had to mock out the environment and refactor the code to enable unit testing. GdxTestRunner and Mockito allowed us to run tests on actual features by directly interacting with the game's core system or by mocking up objects to test.

First, we identified the classes that contained the methods that our requirements covered, and ensured that the states and behaviours are tested. In addition to mocking up objects, we used an environment set up and tear down with GdxTestRunner, to run the game just for the individual test. This allowed us to test the game's main classes directly without refactoring, while still manipulating the game's environment, to confirm the tests cover the necessary method functionality.

To improve code quality and maintainability, we refactored a few classes and abstracted the logic away from the User Interface. This optimised the testing for the refactored methods and improved the structure of the classes. Ideally, by focusing on testing the behaviour of a method, if in future the codebase is refactored, the tests should still be valid. All of our unit tests are testing small parts of our system, and they are fully deterministic, with a clear feedback message regarding the outcome of the test. They are all built to check whether a method is working as expected.

<u>Automated Tests</u>

Our automated tests mainly cover logic heavy requirements, such as achievements, scoring and the leaderboard system, as well as basic game logic such as player movement and avatar selection. Tests are as flexible as possible and are not liable to break if code is tweaked, and there are several replicas of a given test for testing boundaries of legal/illegal inputs. All of our tests run on Ubuntu, Windows and Mac OS.

Test Name	Pass/ Fail	Test Description	Requirement(s)
correctTutorialString	Pass	Checks whether the tutorial displayed at the start of the game is correct or not.	FR_INTRODUCTION
gameEndsAfterSevenDays	Pass	Checks whether the game over screen will be constructed after the player has completed 7 days.	FR-WEEK
eatingGivesBonusScore	Pass	Checks if the bonus scores applied at the end of the game for eating apply correctly.	FR-POINTS-REWARD
recreationalGivesBonusScore	Pass	Checks if the bonus scores applied at the end of the game for recreation apply correctly.	FR-POINTS-REWARD
testPlayerMoveLeft/Right/Up/Down	Pass	4 tests - Checks whether the player can move correctly in the 4 directions.	FR-NAVIGATE
piazzaRandomEnergy	Pass	Tests that a random amount of time will pass when talking to friends at piazza	FR-RANDOM-EVENTS
attemptActivityTooTired	Pass	Tests player won't be able to do activity if not enough energy	FR-SLEEP-ENERGY
attemptActivityTooLate	Pass	Tests can't do activity if past time constraints	FR-SLEEP-TIME
hustleGameInitialises	Pass	Tests game is initialised without crashing	FR-SLEEP-ENERGY
correctResolution	Pass	Tests HustleGame width is set to 1920-1080	FR-RESOLUTION
leaderboardFileExists	Pass	Tests if 'leaderboard.csv' exists in the assets folder	FR_POSTGAME_LEADERBOARD
tenOrLessEntries	Pass	Tests the length of the leaderboard - should be 10 or less. Also tests if the data is formatted incorrectly and provides a different message if this is the case.	FR_POSTGAME_LEADERBOARD
addToLeaderboardLowScore	Pass	Attempts to add a score of 0 to the leaderboard. This should be invalid for a full leaderboard or one with sensible scores.	FR_POSTGAME_LEADERBOARD
addToLeaderboardHighScore	Pass	Attempts to add a score of 1000 to the leaderboard. Should succeed.	FR_POSTGAME_LEADERBOARD

canStudyAtCompSciBuilding	Pass	Tests if the player can study at the CompSci building. Verifies the effects on energy, study hours, and game state.	FR-INTERACT-TEXT, FR-GAME-PLAY-STUDY
canEatAtRonCookeHub	Pass	Tests if the player can eat at the Ron Cooke Hub. * Verifies the effects on energy and game state and dialogueBox text.	FR-INTERACT-TEXT
canGoToSleep	Pass	* Tests if the player can go to sleep. * Verifies the effects on game state.	FR-INTERACT-TEXT, FR-GAME-PLAY-SLEEP
testSelectAvatar1	Pass	* Tests selecting the first avatar. * Verifies that the selected avatar is correctly set to 1.	FR-START
testSelectAvatar2	Pass	Tests selecting the second avatar. Verifies that the selected avatar is correctly set to 2.	FR-START
canChooseToCatchUpInStudies	Pass	Tests if the player can choose to catch up on studies. Verifies the effects on energy, study hours, and game state.	FR-GAME-PLAY-STUDY
canChooseNotToCatchUpInStudie	Pass	Tests if the player can choose not to catch up on studies. Verifies the effects on energy, study hours, and game state.	FR-GAME-PLAY-STUDY
canGoFishing	Pass	Tests if the player can go fishing. Verifies the effects on energy, recreational hours, and game state and dialogueBox text.	FR-GAME-PLAY-RECREATIONAL, FR-ACTIVITY-TIME
canFeedDucks	Pass	Tests if the player can feed the ducks. Verifies the effects on energy, recreational hours, and game state and dialogueBox text.	FR-GAME-PLAY-RECREATIONAL, FR-ACTIVITY-TIME
canMeetFriendsAtPiazzaAndTalkAboutCats	Pass	Tests if the player can meet friends at Piazza and talk about cats. Verifies the dialogue displayed and energy changes.	FR-GAME-PLAY-RECREATIONAL, FR-ACTIVITY-TIME
canClassifyAsBestFisherWhenEnoughFishHaveBeenCaught	Pass	Tests if a player is classified as Best Fisher when enough fish have been caught to classify as a streak, greater than 5.	FR-GAME-PLAY-RECREATIONAL, FR-COUNTER FR-ACHIEVEMENTS

canClassifyAsDuckDuckGoWhenEnoughDucksHaveBeenFed	Pass	Tests if a player is classified as Duck Duck Go when enough ducks have been fed, greater than 5.	FR-GAME-PLAY-RECREATIONAL, FR-COUNTER FR-ACHIEVEMENTS
canClassifyAsBookWormWhenEnoughHoursHaveBeenStudied	Pass	Tests if a player is classified as a BookWorm when enough hours have been studied.	FR-GAME-PLAY-RECREATIONAL, FR-COUNTER FR-ACHIEVEMENTS
cannotClassifyAsBookWormWhenNotEnoughHoursHaveBeenStudied	Pass	Tests if a player is not classified as a BookWorm when they have not studied enough.	FR-COUNTER FR-ACHIEVEMENTS
canInteractWithNPC1	Pass	Tests if the player can interact with NPC1. Verifies the dialogue displayed.	FR-NPCS
canInteractWithNPC2	Pass	Tests if the player can interact with NPC2. Verifies the dialogue displayed.	FR-NPCS

In total we have 41 automated unit tests, with all of them passing.

Manual Tests

Our 37 manual tests cover the rest of our requirements, particularly the ones that require the UI or cover multiple parts of the system as it is difficult to write automated end to end tests for these. Overall, our tests cover everything that was unable to be tested automatically, and sufficiently test the completeness, correctness and appropriateness of all our software components. These are linked on the website.