ENG1 Assessment 2 Report

Cohort 3, Group 23

Deliverable #4: Testing

Oliver Dixon
od641@york.ac.uk

George Cranton
gc1263@york.ac.uk

Praj Dethekar
pd910@york.ac.uk

Denys Sova
ds1855@york.ac.uk

Shivan Ramharry
sr1907@york.ac.uk

Albara Shoukri
aams508@york.ac.uk

Rafael Duarte
rd1395@york.ac.uk

Department of Computer Science
University of York

Semester 2, 2023/24

Overview of Testing Methods

We used automated testing wherever we could, as we believed that maximising coverage and isolating problems to specific areas could be done significantly easier using unit testing, not to mention being much faster than manual testing.

Following advice from academic material released by our professors, we found the testing framework at https://github.com/TomGrill/gdx-testing to be perfectly suitable for our inherited project. This framework could be inserted into any LibGDX game and allowed for “headless” testing of core game functionality, greatly speeding up the time taken for tests to execute. While setting up the headless functionality took us a little time to setup and debug due to considerable code coupling instantiating unnecessary objects, as we planned to write tests for maximum code coverage, the fact that tests could be run without having to instantiate the whole graphics library reduced wasted time greatly.

Furthermore, as it was released under the Apache 2.0 Licence, there were no licensing issues in integrating it directly into our game.

Our automated tests were initially created to test for common game functions that every game necessarily has, such as movement and ensuring the assets for the game exist, then extended for more fine grained targeting towards specific requirements, such as music, interaction, scoring and achievements.

However, certain tests still had to be performed manually, as the requirements they fulfilled were either too qualitative to be tested automatically, or they could be performed much more easily by observing cause and effect. Also, while we did refactor a fair amount of the code to allow automated cases to run, certain tests would fail upon start, as they required the graphics library to be instantiated to be able to test them, thus making manual testing on them mandatory.

Manual tests were derived logically from requirements, where the process of testing them depended on what the requirements demanded. We would make a note of expected behaviour, or for qualitative cases, what the “best” outcome was, and compare that to the output of what we had. For example, for `UR_INTERFACE`, we would start the game and check that all the hud elements were visible, and updated as the game progressed.

The manual tests covered all requirements that the automated tests missed, allowing us to show that each requirement was tested for. While no Functional Requirement was directly tested for, Functional Requirements rely on User Requirements to be fulfilled, and all User Requirements were fulfilled, so we felt it would be superfluous to include them.

Tabulated Test Report

Our automated-testing strategy is primarily composed of six major components: Assets, Audio, Movement, Interaction, Achievement and Score. All automated tests were performed on game functions and classes, asserting internal states.

Automated Tests

Test ID	Requirement(s) Satisfied	Test Name	Test Details	Pass/ Fail
1	NFR_SYSTEM_SIZE UR_MAP_DESIGN	tilemapAssetsExist	Checks if the asset for the main map exists	Pass
2	NFR_SYSTEM_SIZE	fontAssetsExist	Checks if the main font asset exists	Pass
3	NFR_SYSTEM_SIZE	soundAssetsExist	Checks if all game sounds exist	Pass
4	NFR_SYSTEM_SIZE UR_CONTROLS	controlsScreenTexturesExist	Checks if all the textures of the "Controls" screen exist	Pass
5	NFR_SYSTEM_SIZE	menuScreenTexturesExist	Checks if all the textures of the Main Menu exist	Pass
6	NFR_SYSTEM_SIZE UR_INTERFACE	gameScreenTexturesExist	Checks if all the main game assets exist	Pass
7	NFR_SYSTEM_SIZE UR_STUDY_GAME	minigameScreenTexturesExist	Checks if all the minigame assets exist	Pass
8	NFR_SYSTEM_SIZE UR_ENDING	endScreenTexturesExist	Checks if all the textures of the end screen exist	Pass
9	NFR_SYSTEM_SIZE	settingsScreenTexturesExist	Checks if all the textures in the settings screen exist	Pass
10	NFR_SYSTEM_SIZE UR_CHARACTER	playerTexturesExist	Checks if all player textures exist	Pass
11	UR_MUSIC	musicIncrEndsAtMax	Ensures that the music volume can't go past its maximum	Pass
12	UR_MUSIC	musicDecrEndsAtMin	Ensures that the music volume can't go past its minimum	Pass
13	UR_MUSIC	soundIncrEndsAtMax	Ensures that the game sounds volume can't go past its maximum	Pass
14	UR_MUSIC	soundDecrEndsAtMin	Ensures that the game sounds volume can't go past its minimum	Pass
15	UR_MUSIC	upSoundToggle	Ensures that the "up" sound turns off if activated again	Pass
16	UR_MUSIC	downSoundToggle	Ensures that the "down" sound turns off if activated again	Pass
17	UR_MUSIC	buttonSoundToggle	Ensures that the button press sound turns off if activated again	Pass

18	`UR_MUSIC`	`eatingSoundToggle`	Ensures that the eating sound turns off if activated again	Pass
19	`UR_MOVEMENT`	`testMoveLeft`	Checks player can move left	Pass
20	`UR_MOVEMENT`	`testMoveRight`	Checks player can move right	Pass
21	`UR_MOVEMENT`	`testMoveUp`	Checks player can move up	Pass
22	`UR_MOVEMENT`	`testMoveDown`	Checks player can move down	Pass
23	`UR_INTERACTION`	`testDoorCollision`	Ensures collision with doors happens where player touches it	Pass
24	`UR_INTERACTION` `UR_EATING`	`testEatButton`	Ensure the eat button works when clicked	Pass

25	`UR_INTERACTION`
	`UR_STUDYING`
	`UR_TIME_SKIP`
		`testStudyButton`	Ensure the study button works when clicked	Pass

26	`UR_INTERACTION`
	`UR_RECREATION`
	`UR_TIME_SKIP`
		`testExerciseButton`	Ensure the exercise button works when clicked	Pass

27	`UR_INTERACTION`
	`UR_SLEEPING`
	`UR_TIME_SKIP`
		`testSleepButton`	Ensure the sleep button works when clicked	Pass

28	`UR_ACHIEVEMENTS`	`testAchievementInitialisation`	Ensure achievements are initialised properly	Pass
29	`UR_ACHIEVEMENTS`	`testModifiedName`	Ensure every level of achievement works properly	Pass
30	`UR_PLAYER_SCORE`	`testScoreUpdate`	Ensure score calculation works properly	Pass
31	`UR_PLAYER_SCORE`	`testFinalUpdate`	Ensure final score calculation with achievement bonuses works properly	Pass
32	`UR_PLAYER_SCORE`	`testScoreReset`	Ensure score multipliers are reset when needed	Pass
33	`UR_TIME_SCALE`	`testTimeScale`	Ensure passage of time is normal	Pass

When our automated tests failed we were easily able to narrow down and pinpoint the root cause of the problem and fix it, and thus, we were able to ensure that every one of them passed in the end. The manual tests were performed individually upon each of the missed requirements, categorised into User Requirements and Non-Functional Requirements.

Manual Tests

Test ID	**Requirement Tested**	**Why Manually Tested**	**How Manually Tested**	**Pass/ Fail**
34	`UR_INTERFACE`	Game has to be fully instantiated to reveal interface	Main game screen loaded and User Interface checked	Pass
35	`UR_CHARACTER`	Character is easiest seen visibly swapped	Character selected in settings and main game loaded	Pass
36	`UR_CONTROLS`	Explained through a tutorial	Controls screen opened through main menu	Pass

37	UR_ACCESSIBILITY	Subjective opinion	Game playtested for user evaluation by various students	Pass
38	UR_GAME_LENGTH	Cannot be checked without playing the game	Time taken to complete recorded of various students	Pass
39	UR_MAP_DESIGN	Map is visibly seen	Game loaded and map checked	Pass
40	UR_CAMPUS_BUILDINGS	Labels are visibly seen	Labels on buildings checked	Pass
41	UR_NAME	Name shown on screen at end	Game played till end	Pass
42	UR_LEADERBOARD	Leaderboard shown on screen at end	Game played till end	Pass
43	UR_ENDING	Ending shown at end	Game played till end	Pass
44	UR_MINI_GAME	Game must be interactively played	Both minigames played and checked for abnormalities	Pass
45	UR_GAME_OVER	Game graphics library must be instantiated	Game played till end	Pass
46	UR_PAUSE	Game graphics library must be instantiated	Esc pressed on main game screen	Pass
47	UR_OBJECTIVE	Objective visibly seen	Controls screen opened through main menu	Pass
48	UR_BACKSTORY	Backstory visibly seen	Controls screen opened through main menu	Pass
49	NFR_JAVA_VERSION	Game coded in Java 11	Game compiled to jar in Java 11	Pass
50	NFR_COMPATIBILITY	Not an automatable task	Jar opened on all 3 OSes and game played till end	Pass
51	NFR_SYSTEM_SIZE	Size given after compilation	Game jar file size checked	Pass
52	NFR_PERFORMANCE	Keypresses are user input	Lag to be felt by user evaluation	Pass
53	NFR_SCALABILITY	Depends on game architecture	Architecture checked for coupling	Fail
54	NFR_MAINTAINABILITY	Depends on code commenting	Code checked for javadocs	Pass
55	NFR_RELIABILITY	Game needs to be run for long times	Game kept on menu for 50 hours, also multiple playthroughs tested	Pass
56	NFR_COMPLIANCE	Need to ensure licences are adhered to	Licences of each module and asset used checked	Pass
57	NFR_EFFICIENCY	Resource utilisation needs to be monitored while game fully runs	Game run fully and impact on performance of computer checked	Pass

While most tests passed as they were assertions of truth and collective opinion, one test, `NFR_SCALABILITY` was not able to pass, as we had significant difficulty implementing our own features into the game and refactoring the entire codebase would be beyond the scope of the assessment, thus making our collective opinion that `NFR_SCALABILITY` could not be fulfilled. However, as most of the other manual tests passed, we were happy with an overall 98% test pass rate, and were able to cover all the requirements of the game.

Generated Reports and Analyses

Automated Testing

The automated coverage report, generated by the JetBrains IntelliJ coverage tool, can be found at https://eng1-group-23.github.io/A2-website/html-coverage-report/. The automated tests report, also generated by the JetBrains IntelliJ J-Unit integration tool, can be found at https://eng1-group-23.github.io/A2-website/test-results.html

Manual Testing

Most of the manual tests, like `UR_INTERFACE` or `UR_CHARACTER` were assertions of truth which could not be automatically tested, and as such, to test them, we would simply play through the game till the point at which they would be triggered.
For collective opinion tests that didn't require heavy emphasis on fairness, such as `NFR_SCALABILITY` or `NFR_MAINTAINABILITY`, we would use tools to automatically generate reports, such as IntelliJ's UML diagram or Javadocs HTML generator, and browse through to see if code was sufficiently documented and decoupled.
For subjective opinion, such as `UR_ACCESSIBILITY` or `NFR_PERFORMANCE`, we decided that the fairest opinion would be that of the playtesters. We got the opinions for these tests from our playtesters, and if their general consensus was that the requirement was achieved, we marked it as a pass.