Test Plan

0. Introduction

We aim to use automated testing where possible, including to test utilities such as timer, score and point. Where this is not appropriate, manual testing will be carried out, including for navigation around menus in gameplay, and map rendering. We will use exploratory testing throughout manual testing, with the aim of higher coverage. Testing for failure scenarios will be carried out throughout.

We aim to describe how we will test each requirement, describing the mode of testing, and the expected outcomes. Some requirements will require both automated and manual tests, such as UR_EVENT, and as such will be included in both sections. In further documents, we will extend this to include the scope of the input, the coverage and outcome of the tests, and whether any further testing was required. Any requirements that were felt to be not testable in the normal sense will be also discussed here.

A traceability matrix will be constructed towards the end of the project, for a clear representation of how and where each requirement was tested.

1. Automated Testing

We will use the JUnit framework to write our automated tests for the project. Most will use only unit tests, with some interaction testing being used for events and achievements. Test doubles using the LibGDX headless backend will be required for more complex tests, such as those concerning events and achievements.

Jack will write the tests, with Aidan carrying out analysis in line with the work on Continuous Integration. These tests will be executed every time new code is pushed to the main branch, ensuring faulty code can be found and corrected immediately. Developers should then ensure that they are maintaining different branches correctly (and should push completed code to main regularly), in order to avoid excess development on top of untested modules.

These requirements can be quantitatively interpreted, indicating they would be suitable for automated testing.

Req. ID	Test ID	Test Description
UR_EVENTS	1.1	The functionality of the event and eventManager classes will be tested. - type of event - attributes of event (testing getters and setters) - how an event changes with time - the effects of an event
UR_STUDENT_SATISFACTION, FR_SCORE	1.2	Test the functionality of the score and scoreManager classes.

		- basic methods of score class - how score is affected by other areas
UR_GAME_PROGRESS	1.3	Test basic functionality of the timer utility.
UR_ACHIEVEMENTS, FR_ACHIEVEMENTS	1.4	Test functionality of the achievement and achievementManager classes. - creating a new event - attributes of the achievement - methods relating to achievements - how are they achieved, when and where are they displayed
FR_MAP	1.5	Test the availability of the map asset.
FR_BUILDING	1.6	Test the availability of the building assets

A short note on testing the consistency and correctness of the score algorithm: The algorithm for calculating the student satisfaction score related to building placement is planned to be dependent on a combination of variance and density. That is, the user will want to keep the variance of the numbers of different types of buildings to a minimum, by placing similar numbers of different types of buildings. The player will want to have a high density of buildings around each building.
These will be implemented as separate functions, making for good testability. Testing of these specific functions should be done as part of automated testing.

2. Manual Testing

We plan and carry out manual testing for the remaining requirements thought to need such testing.

These tests will be carried out throughout the technical implementation of the project, in line with the development of different modules. Towards the end of the project, they will be carried out and formalised in a more detailed manner, with a full description of the scope and aim of every test written. This will be done by Erin.

Manually testing these requirements was thought to be the best option, as a result of a heavy reliance on either rendering, or on ease of gameplay/navigation.

Req. ID	Test ID	Test Description
UR_EVENTS, FR_EVENTS	2.1	Through gameplay, can the user interact (see and react to) events occurring. - is the event displayed - does the event have an affect on the state of the game - can the user react to the event in an appropriate manner - does the user reaction have an affect on the state of the game

UR_STUDENT_SATISFACTION	2.2	Can the user interact with the game in a way that changes the student satisfaction. - can the user interact to increase the score - can the user interact to decrease the score - can the user interact inside of events to increase the score - can the user interact in order to win achievements increasing the score
UR_GAME_PROGRESS, FR_TIME_LIMIT, FR_TIMER, NFR_END_OF_GAME	2.3	Does the game convey a sense of time passing within the game, and outwards to the user. - is a timer displayed showing the amount of time before the end of the game - does the gameplay change over time in a way that emulates time passing within the game - does the event timer stop/play when the game timer is stopped/played - does the game end at the end of the timer
UR_BUILDING_COUNTER	2.4	Is there a counter showing how many of each type of building has been placed.
UR_LEADERBOARD, FR_LEADERBOARD	2.5	- can the user enter, save, and change a username - is the username then saved to a leaderboard with the user's score - is the leaderboard displayed at the end of the game
UR_ACHIEVEMENTS, FR_ACHIEVEMENTS	2.6	Does the user receive achievements based on their interactions? - can the user interact in such a way that means an achievement is awarded? - does the system always recognise the achievement - is the achievement displayed to the user in an understandable way
FR_MAP, UR_CAMPUS_CREATION	2.7	Does the game provide a visual map
FR_BUILDING, UR_CAMPUS_CREATION, UR_BUILDING_VARIETY	2.8	Can the user place buildings on the map - can the player determine where the building is placed - does the building then remain placed there
FR_BUILDING_TYPES, UR_BUILDING_VARIETY	2.9	Can the user place multiple types of different buildings? - can the user pick a building type and then place it - can the user pick a building type and then pick a different building type, and the second type is placed
FR_SATISFACTION_BUILDINGS	2.10	Does the score change based on the number of buildings placed - can the user place buildings in a way that will increase the score - can the user place buildings in a way that will decrease the

		score - do buildings being placed close together increase the scoe
FR_OBSTACLES, UR_CAMPUS_CREATION	2.11	Does the map have obstacles - can the user place a building over an obstacle - can the user clearly tell where obstacles are
FR_BUILDING_COUNTER	2.12	- when the user places a building of a certain type, does the counter corresponding to that type increase by 1 - when the user picks up a building and doesnt place it, does the building counter increase
	2.13	Do all the buttons work as normal when the game window is resized.

3. Features not to be tested

UR_IMMERSION, UR_TARGET_MARKET, UR_UX, UR_MAINTAINABIITY, FR_USER_INTERFACE will all be tested as a part of user evaluation, and so it was felt that further formalised testing would be unnecessary.

NFR_PERFORMACE and NFR_INTERACTIVE_ELEMENT_REACTION are not included in the overall testing section, but the program specs will be checked towards the end of the project, to confirm they line up with the requirements.

NFR_OPERABILITY, NFR_IMMERSION and NFR_THEME will all be tested in line with their associated functional requirements in the user evaluation.

NFR_ERROR_MESSAGES, NFR_DOCUMENTATION and NFR_CODE_MODULARITY fall under the responsibility of the development team. A short report on these will be included in the manual testing report.

4. Pass/Fail criteria

Every automated test must pass.

For manual testing, if all test cases that fall under the user requirement priority level of “should” pass, the system will be considered to have passed the manual tests. Otherwise, further development and testing will be needed by the developers.