Software Testing Report

ENG1 Team 9

Jacob Dicken
Bertie Cartwright
William Croft
James Dovener
Henry Chan

Methods and Approaches

For unit testing, we used JUnit5 to enable us to easily write and run short tests that test small portions of our code. We chose JUnit5 for its simplicity and easy integration with Maven, our build system. This is a suitable approach for developing a small game, as we could begin writing tests quickly and efficiently, sacrificing a minimal amount of our limited collective time working on setting up an automated testing solution.

When designing a CI pipeline at the start of development, we set up automated unit testing with JUnit and code coverage report generation with JaCoCo. This allows us to track the effectiveness of the tests we have written and identify new areas of the codebase that need testing. This encourages writing code in a testable manner, allowing for a significant portion of the codebase to be covered by tests. This approach to testing makes future development and refactoring much easier as we can verify that existing code is valid and any changes will be reflected in the next integration - which should be within a short time due to our CI process.

Using the test reports generated by JaCoCo, we can identify which lines and execution paths through the program are tested, identifying potential missed test cases that would not be visible by simply looking at which tests pass/fail. This is helpful for use in code reviews prior to merging a new feature, as missing tests can be identified and added to ensure the code is correct.

We also designed a set of manual end-to-end tests to be carried out in conjunction with these automated tests. The manual tests are structured as a table with columns for the ID, objectives, preconditions, steps to execute and expected result. They are designed to test the gameplay and user interface holistically in a way that is not possible with automated tests. For example, all user interface elements are covered in a manual test, even if they are partially covered in an automated test. This is so we can guarantee that these high-level elements of the program work and to test that all parts of the codebase can function as a complete product.

Test Results and Coverage

All 124 automated tests were passed by the current implementation, both when being run locally and by the Github Actions CI workflow. All 10 manual tests are also passed by the current implementation.

The automated tests are split into categories based on which section of the code they test and cover elements of the following requirements:

Test File	**Requirements Tested**
AchievementTest.java	FR_ACHIEVEMENT_EARN
SatisfactionAchievementTest.java	FR_ACHIEVEMENT_CRITERIA FR_ACHIEVEMENT_REWARDS
BuildingCounterTest.java	FR_BUILDING_COUNTER
BuildingFactoryTest.java	FR_BUILDING_VARIETY FR_BUILDING_MANAGER
BuildingManagerTest.java	FR_BUILDING_MANAGER
BuildingTypeTest.java	FR_BUILDING_VARIETY
MapObjectTest.java	FR_BUILDING_PLACEMENT
ClockTest.java	FR_TIMER_COUNTDOWN FR_TIMER_START FR_GAME_PAUSE_EFFECT
CollisionDetectionTest.java	FR_BUILDING_PLACEMENT FR_OBSTACLES
GridAreaTest.java	FR_BUILDING_PLACEMENT
GridPositionTest.java	FR_BUILDING_PLACEMENT
ScoreManagerTest.java	FR_LEADERBOARD FR_ACHIEVEMENT_REWARDS
LeaderboardTest.java	FR_LEADERBOARD FR_ADD_TO_BOARD
IntRangeTest.java	FR_BUILDING_PLACEMENT
MenuTabTest.java	FR_BUILDING_SELECT FR_BUILDING_MANAGER
DepthSortingTest.java	FR_ISOMETRIC_CAMERA
StudentSatisfactionTest.java	FR_SATISFACTION_BUILDING_DISTANCE FR_BUILDING_RATIOS FR_EVENT_TYPES

The manual tests cover all remaining functional requirements and overlap many of the functional requirements that are tested by the automated testing. They also cover some non-functional requirements. A table detailing which requirements are covered by the manual tests can be found here. As each user requirement has one or more functional or non-functional requirements relating to it, our test coverage implicitly covers each user requirement. The only user requirements that are not covered by any test are UR_USABILITY and UR_EXTENDABILITY. UR_USABILITY relates to the ease of use by a first-time user, which we can't test easily or fairly, even with a manual test. The closest we were able to come to testing this requirement was in our user evaluation. UR_EXTENDABILITY relates to the structure and quality of the code of the game. As such, the closest we were able to come to testing the requirement was in our automated code style report in our CI workflow.

We favoured automated tests for any part of the program that did not involve user input or graphical elements. Automated tests have 36% code coverage, which includes almost all of the game logic. They allow us to have more confidence that these parts of the program work with a variety of inputs and preconditions and do not fall victim to edge cases. These tests were especially helpful for features such as student satisfaction, as they allow us to verify that each factor that contributes to the satisfaction works as intended by testing them in isolated cases.

Manual tests cover all user interface and gameplay elements. This allows us to ensure that the game works as a complete product. When used in conjunction with the unit tests, they allow us to verify that the game will work as the user expects it to across almost every use case.

Testing Material URLs

The manual testing report, code coverage report and code style check reports can be found on the website testing page, with an additional link to the manual testing page provided below.

Website Testing Page: https://jd760.github.io/testing.html
Manual Tests: https://jd760.github.io/diagrams/assessment2/testing/Manual%20Tests.pdf