Testing Report

Group Number: Cohort 2 Team 7

Group Name: pickNmix

Group Member Names:

David Lun

Sameer Minhas

Harry Muir

Phyo Lin

Alex McBride

Our game testing approach follows the software testing guidelines to make sure that the functionalities and quality of the game aligns with the principle requirements of the game from the product brief. This comprehensive approach tests every aspect of the game from the individual methods and components to the overall user experience of the game by utilizing the unit testing and manual testing.

Unit testing forms the foundation of our testing approach. It focuses on verifying the correctness of individual components, such as the calculation of satisfaction scores, the checking the building count, the logic governing building placement,and the handling of game events. We created the thorough test cases using the JUnit framework, using the Given-When-Then structure to ensure the clarity, modularity and traceability of the methods. This stage of testing is essential in order to check the correctness of the isolated methods and identify the bugs of them to fix.

Manual tests are also used for the aspects of the game that cannot easily be tested through the automated tests. We used manual testing for the areas like evaluating the game flow, usability and responsiveness.

Testing approaches like boundary testing and input space partitioning are used to check the accurate handling of edge cases and diverse inputs. These approaches are used for the methods that also accept a specific range of inputs. By employing these approaches for those kinds of methods, we can make sure that our game works well under the unexpected conditions and prevent the failures because of the invalid inputs.

I believe that our testing approach which is combined with both automated testing and manual testing are appropriate for our project because each testing can cover the critical areas of the game that one testing cannot completely cover by itself. Therefore, combining them make sure that we check every aspect of the game without overlooking anything.

Our testing strategy makes sure that our game meets its functional and non-functional requirements while maintaining the quality and reliability of our game.

Test Report

Automatic Tests

We used JUnit to facilitate our testing, every major class in the game applicable for automatic testing. Data classes and enums are not functional and thus do not have tests, as well as anything that cannot be automatically tested, such as the visual aspects of the rendered interface. In addition to JUnit testing we deployed JaCoCo in GitHub Actions to provide coverage reports for our tests, uploading results as an artifact.

All core functionality of the Server package was tested automatically, with only disconnected derivative classes being ignored such as the individual Events and Achievements. Both types extend the Event superclass, and perform minimal logic. The EventHandler, responsible for processing these events is thoroughly tested. The remaining testable components of Server are fully tested such as TimeHandler and Simulation. This aspect of the game was the most important to be tested, given it handles the majority of the game logic and contains the most complex functionality.

All useful functionality within the Broker class was tested automatically, including Popup brokering, leaderboard management, coordinate checking and building handling. Broker is a standalone client/server agnostic singleton class implementing primarily logical methods that can be tested in entirety. Simple methods that do not require testing such as getters and setters were ignored.

The majority of the Entities package of the game cannot be automatically tested as their functionality is either heavily or fully consisting of visual aspects, frequently cannot be detached from libGDX-dependent processes, or lack anything meaningful to test (i.e. getters/setters). Most of the actual logic of these classes include that which facilitates dynamic adjustment of rendered components, and is thus not feasibly testable automatically.

Most of AbstractGameScreen and its child classes cannot be usefully tested, either because the method functionality is visual display or the method is fully dependent on the rendering process to run. Renderer and InputHandler cannot be tested as they are directly tied to the Rendering process, as well as Main.

The aspects of the game that rely on visual aspects cannot be reliably tested, which includes many of the non-functional requirements. There are minimal viable methods to test these with and frequently little use in doing so. User-facing requirements such as ease of use cannot be reliably tested, and alignment and spacing of UI elements are often arbitrary and untestable.

All implemented tests correctly verify important game functionality is as expected and as reflected by the visual aspect of the game. The JaCoCo report showed that our tests covered 38% of the implementation, which was as expected given the aforementioned large amount of untestable code on the client side. Looking closer at the broken down details of this report we can see that 75% of the Broker and 42% of the server side are covered, which is where most of the testable logic is situated, with the missing percentage of the server side consisting of the events and achievements considered unnecessary to test as mentioned.

Test results

Throughout the testing process, the automated tests identified several issues and incorrect logic within the code, both initially after their writing and over time in response to how the code evolved.

The first iteration of BrokerBuildingCountTest identified an issue with the overload of Broker's getBuildingCount method, getBuildingCount(Building building), which reported still only one instance of the supplied building after a second building of that type had been added to the internal game state. The identified issue was caused by the incorrect handling of buildings in the internal state, previously mapping the integer count to each individual instance of each building type rather than the types themselves. Following the discovery of this via the test, the addBuilding(Building building) method was reworked to function as intended, and in the process was updated to use the more modern handling of building types throughout the code, which had previously also been missed.

During the implementation of TimeKeeper, a test in TimeKeeperTest identified an issue with the accessing of stored time for the currentGameTime method where if the time was set to 0, before the TimeKeeper was started, the method would evaluate to negative UNIX time. This was then patched to return the value of -1, which is never used otherwise, and react to receiving this value by correctly returning 0. This test case would likely not have been ever detected easily by manual testing, and have been difficult to identify without the unit test.

The popup brokering system between client and server in Broker correctly identified an issue with the handling of PopupTickets by client. The original implementation removed the top of the queue of pendingPopups from Broker's object and handed it completely to client, which was then inserted back into Broker's resolvedPopups queue. This resulted in issues identified by the tests, before the full functionality of popups had been finished and the issue had an opportunity to arise organically, in keeping track of tickets and potential duplication of PopupTicket instances being handled by the game. Rewriting this to use the new system, where pending and resolution is entirely handled internally and client can only peek the queue also made it far easier for server's implementation to keep track of the tickets it had pushed to client. Due to scope of the requirements for the implementation, the server never has a need to retrieve responses from PopupTickets in the current game, however the framework for arbitrary number of user selectable options pushed and retrieved from the server side would be fully functional.

There was one test in PopupManagerTest that did not succeed that was eventually disabled as the scope of PopupManager had shrunk to a less significant role in the popup process and manual testing confirmed that the intended functionality was actually performing correctly. The cause of the test's issue was not fully identified but it was evident that the test itself was broken rather than the code it was intended to test. The cause of this misbehaviour was not identified but the methods called in PopupManager are essentially server side wrappers for the popup-related methods within Broker, which are fully tested and confirmed to be working as intended by BrokerPopupTest.

(Non-functional requirements excluded - Automatic testing not possible)

UR_CAMPUS_CONSTRUCTION	BrokerBuildingTest.placeBuilding
UR_BUILDING_VARIETY	BuildingFactoryTest.buildingVariety
UR_BUILDING_COUNT	BrokerBuildingCountTest.getTotalBuildingTest
UR_PAUSE_FUNCTIONALITY	TimeKeeperTest.pausePausesAndUnpauses
UR_LEADERBOARD	BrokerLeaderboardTest
UR_ACHIEVEMENTS	Manual
UR_MOVE_BUILDINGS	BrokerBuildingTest.{placeBuilding, destroyBuilding}
UR_TIME_ELAPSED	TimeKeeperTest.roughlyOneSecondPassed
UR_MUTE	N/A
UR_SOUND_EFFECTS	N/A
UR_BUILDING_RESTRICTIONS	BrokerCoordinateTest
UR_REMOVE_BUILDINGS	BrokerBuildingTest.destroyBuilding
FR_DISPLAYED_BUILDINGS	Manual
FR_BUILDING_VARIETY	BuildingFactoryTest.buildingVariety
FR_BUILDING_SELECTION	Manual
FR_BUILDING_PLACE	BrokerBuildingTest.placeBuilding
FR_BUILDING_COUNT	BrokerBuildingCountTest.getTotalBuildingTest
FR_PAUSE	TimeKeeperTest.pausePausesAndUnpauses
FR_MOVE_BUILDINGS	BrokerBuildingTest.{placeBuilding, destroyBuilding}
FR_BUILDING_RESTRICTIONS	BrokerCoordinateTest
FR_MUTE_FUNCTIONALITY	N/A
FR_DISPLAY_TIME	TimeKeeperTest.roughlyOneSecondPassed
FR_SOUND_EFFECTS	N/A
FR_REMOVE_BUILDINGS	BrokerBuildingTest.destroyBuilding
FR_LEADERBOARD	BrokerLeaderboardTest
FR_ACHIEVEMENTS	Manual

Testing Material - Testing Results and Coverage Report

https://grumv2.github.io/Eng1Ass2Web/WebDocs/pdfs/jacoco-report/index.html

Manual Testing

https://grumv2.github.io/Eng1Ass2Web/WebDocs/pdfs/manual_testing.pdf