Assessment 2: Brownfield Development - Testing

Cohort 1 Team 2

Team Members:

Trace Chinelle
Vidhi Chohan
Apollo Cowan
Siyuan Liu
Aryaman Marathe
Charlie Mason

4. Software Testing Report

a) Our team opted to use JUnit as our testing framework along with Mockito for mocking classes. The jacoco library is used to generate a code coverage report and gradle's build system tooling allows for an html testing report to be generated. JUnit works well with libGDX because libGDX uses gradle which provides easy ways to integrate JUnit and jacoco. Introducing Mockito is also simple which decreases developer time and allows us to focus on the code itself rather than spend large amounts of time on configuring the build system.

Components of the project that are decoupled from rendering are automatically tested using JUnit. However, our project relies heavily on manual tests due to the nature of the testing libGDX applications, using headless mode provided by libGDX did not yield any difference in code coverage. Components of the codebase related to rendering are tightly integrated with program logic, hence integration tests for most components require a graphical context and a full-fledged desktop environment to be available. Although manual testing is hard to scale and is time consuming, it is far more approachable than spending developer time on automating the testing of code relying on OpenGL's graphical context to exist.

b)
Manual Test Report

In depth Manual Test Descriptions: https://areamanm.github.io/ManualTesting.html

Test ID	Description	Category	Requirements	Actual Outcome	Status
MT_001	Building purchase and placement validation	Functional Testing	FR_BUYING	Buildings that cost less than the balance can be selected and placed. Otherwise cannot be placed. Correct amount deducted from balance once a building has been purchased.	Pass
MT_002	Game timer and summary screen	Functional Testing	FR_TIME_LIMIT	Game over screen is displayed once the timer reaches 0. Final satisfaction score, leaderboard, and any unlocked achievements are displayed. No further game actions are possible.	Pass
MT_003	Obstacle building placement prevention	Functional Testing	FR_OBSTACLES	Building placement is prevented on all obstacle types. Visual indicator shows invalid placement.	Pass
MT_004	Achievement Menu	Interface Testing	FR_ACHIEVEMENT_MENU	There is no achievement menu.	Fail
MT_005	Performance Testing	Performance Testing	UR_PERFORMANCE	Game ran without lags, crashes, and ran for an extended period on a medium-spec computer, even when multiple actions were made rapidly.	Pass
MT_006	UI Intuitiveness	Usability Testing	UR_INTUITION	Easy to understand notifications. Has intuitive controls and clear UI indicators for building placement. Elements are self explanatory. Except the play button must be clicked to begin the game. This is not stated in the tutorial.	Pass
MT_007	Visual Consistency	UX testing	UR_IMMERSION	Art style is consistent. Visual theme is cohesive. Visual feedback is clear.	Pass
MT_008	Game progression	Functional Testing	UR_GAME_PROGRESS	5 minute timer works correctly and accurately. Years progress appropriately. Game over screen displays correct statistics.	Pass

MT_009	UI information display	Interface Testing	FR_USER_INTERFACE	Only essential information is shown. Display is clear and readable. The amount of information is not overwhelming. Important information is prominent.	Pass
MT_010	Game Timer Display	Interface Testing	FR_TIMER	Timer is clearly visible in the infobar. Year number updates correctly. The time remaining is accurate.	Pass
MT_011	Leaderboard Display	Functional Testing	FR_LEADERBOARD_MENU	The top 5 scores are visible. The scores are in descending order.	Pass

Test MT_004 failed. This was because instead of having a clickable button that took the user to the achievements screen, the unlocked achievements are displayed on the game over screen. We chose to do this to streamline the user experience by immediately highlighting their achievements without requiring an additional interaction. This will not be passed due to time constraints and the fact that it is a minor error as users can still see which achievements they have unlocked at the end of each game on the game over screen.

Test MT_006 partially failed. There is one action at the start of the game that is not immediately intuitive. Specifically, the play button must be clicked to begin the game, but this step is not explicitly stated in the tutorial or made noticeable to players visually. While the majority of features are intuitive and function as intended, this oversight may cause minor confusion for new players. This can be corrected by adding a statement in the tutorial, for example “Click the play button to start!”.

All other tests passed, this was because a lot of the UI was done by team 9 and by building upon it we could improve upon it. This made the tests in the categories interface testing and UX testing fairly easy to pass as we had an idea of how the game roughly looked before we took on team 9’s project in this assessment.

Automated Test Report

Test ID	Requirements tested	Description	Outcome
AT_001	UR_ACHIEVEMENTS	Verify that achievements are loaded correctly by the Achievements class and marking achievements	Pass
AT_002	UR_EVENTS	Verify that the first event is always the tutorial event and	Pass

		that every even event is additional funding for the user	
AT_003	UR_LEADERBOARD, FR_NEW_HIGH_SCORE	Verify that the Leaderboard class correctly orders high scores while accepting new scores and formats scores in the way required for the UI to correctly display them	Pass
AT_004	FR_TIME_LIMIT, FR_TIMER	Verify that the timer class correctly ticks, updates the current event number and signals the end of the timer after the input time has passed	Pass
AT_005	-	Verify that the Point class returns a valid string representation and has a working equals operator	Pass

There are 5 test suites containing 16 methods that run tests related to a single class used in the project: A test for the achievements class, timer class, leaderboard class, event class and the point class.

The Point class is a crucial component used throughout the codebase for dealing with the grid structure of the in-game map and while it does not directly correspond to a requirement, it is a key component of meeting requirements related to buildings such as FR_BUILDING and FR_OBSTACLES and hence is tested to ensure correctness.

All 16 tests in the automated test suite passed, however a large part of the codebase could not be automatically tested due to key components of the codebase relying on an OpenGL context being available with access to game assets and a graphical user interface. More extensive planning could have allowed for further decoupling of program logic from rendering code and allowed for more automatic testing of key components of the codebase, however, code that relies on libGDX's rendering capabilities is difficult to reliably test across the very wide range of possible hardware configurations the game can be played on. It was clear from the initial stages of the project that testing the UI and performance of the game is going to heavily rely on manual tests.