SOFTWARE TESTING

REPORT

Group 3

Liam Martin

Aaliya Williams

Lucy Crabtree

Kai Nichol

Sammy Hori

Tim Gorst

Zac Ribbins

A) Testing Methods and Approaches

In our project, we adopted a blend of automated and manual testing methodologies tailored to our system's specific needs. This approach aimed to ensure thorough coverage while maintaining a balance between efficiency and accuracy.

Automated Testing
Automated testing formed a critical aspect of our quality assurance strategy, ensuring the reliability and robustness of our software. We meticulously crafted and executed 49 automated tests, covering various aspects of the system's functionality. These tests were organised into different packages and classes, facilitating targeted validation of specific modules and functionalities.

Initially, we established unit tests primarily to support continuous integration, ensuring that inadvertent code changes didn't compromise existing functionality. These tests were automated using GitHub Actions, triggered by commits to the master branch or pull requests. While adjustments were occasionally necessary to accommodate code changes, these tests served as a crucial safety net against regression.

Certain unit tests were designed to directly address functional requirements, such as PlayerMovementUnitTests. These tests partially addressed UR_CONTROLS and FR_CHARACTER_MOVEMENT by verifying correct player velocity behaviour when specific keys were pressed and released. This approach helped ensure that the codebase adhered to specified requirements from the outset, promoting robustness and clarity in implementation.

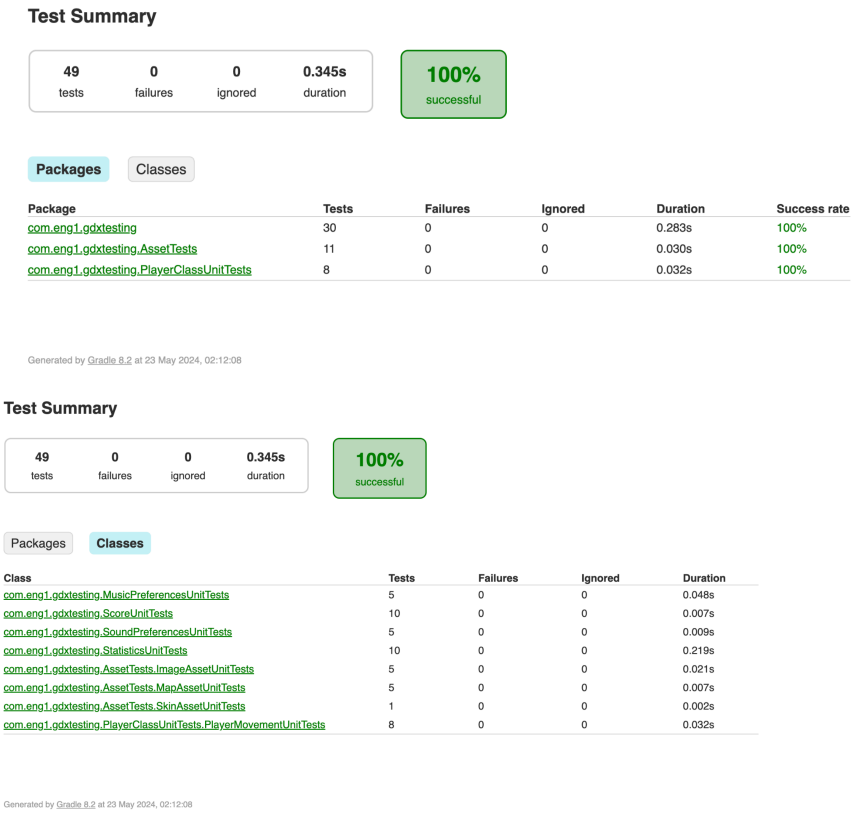
Manual Testing
Recognising the dynamic nature of our development process, we extensively employed manual testing. This involved crafting a range of manual tests to cover all functional requirements of the game. These tests were regularly executed on the master branch to monitor progress and were pivotal in validating changes made in feature branches.

Manual testing provided invaluable insights by ensuring that users experienced expected outcomes firsthand, transcending mere code-based validation. This approach guaranteed the integrity and usability of the system from an end-user perspective.Also, manual tests for unimplemented features served as an informal TDD mechanism, delineating expected behaviours and aiding in the validation of forthcoming functionalities.

B) Test Report

Our testing efforts encompassed a variety of unit and manual tests aimed at validating both continuous integration and feature development aspects. Unit tests were employed to verify core functionalities and were automatically executed through GitHub actions. Manual tests, on the other hand, were extensively used to ensure comprehensive coverage of functional requirements and validate user experiences.

The test coverage report, available at https://sammyhori.github.io/eng1-part-2-website/, showcases our automated testing efforts.



Test Summary

49 tests	0 failures	0 ignored	0.345s duration	100% successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
com.eng1.gdxtesting	30	0	0	0.283s	100%
com.eng1.gdxtesting.AssetTests	11	0	0	0.030s	100%
com.eng1.gdxtesting.PlayerClassUnitTests	8	0	0	0.032s	100%

Generated by Gradle 8.2 at 23 May 2024, 02:12:08

Test Summary

49 tests	0 failures	0 ignored	0.345s duration	100% successful

Packages | Classes

Class	Tests	Failures	Ignored	Duration
com.eng1.gdxtesting.MusicPreferencesUnitTests	5	0	0	0.048s
com.eng1.gdxtesting.ScoreUnitTests	10	0	0	0.007s
com.eng1.gdxtesting.SoundPreferencesUnitTests	5	0	0	0.009s
com.eng1.gdxtesting.StatisticsUnitTests	10	0	0	0.219s
com.eng1.gdxtesting.AssetTests.ImageAssetUnitTests	5	0	0	0.021s
com.eng1.gdxtesting.AssetTests.MapAssetUnitTests	5	0	0	0.007s
com.eng1.gdxtesting.AssetTests.SkinAssetUnitTests	1	0	0	0.002s
com.eng1.gdxtesting.PlayerClassUnitTests.PlayerMovementUnitTests	8	0	0	0.032s

Generated by Gradle 8.2 at 23 May 2024, 02:12:08

Our automated testing regimen played a pivotal role in our quality assurance strategy, ensuring the robustness and reliability of our software project. A total of 49 automated tests were created and executed, covering diverse aspects of the system's functionality. These tests were organised into different packages and classes, facilitating targeted validation of specific modules and functionalities.

All automated tests passed successfully, affirming the efficacy of our testing efforts and the stability of the codebase. The comprehensive test suite encompassed various categories, including asset validation, player class functionalities, sound and music preferences, score calculation, statistics management, and player movement behaviours.

The automated testing process demonstrated efficiency, with a total execution duration of 0.345 seconds. This rapid turnaround time ensured timely feedback on code changes, facilitating continuous integration and deployment practices. Moreover, the absence of test

failures or ignored tests underscores the reliability and consistency of the automated testing framework.

Automated testing played a crucial role in mitigating the risk of regression and ensuring the integrity of the codebase throughout the development lifecycle. By systematically validating code changes against predefined test cases, automated testing bolstered confidence in the software's functionality and helped maintain high-quality assurance standards.

While unit tests contributed to the stability of our codebase and provided reassurance against inadvertent regressions, manual tests were indispensable in validating the user-facing aspects of the application. The decision to prioritise manual testing was validated by the frequent and significant changes observed in the codebase, necessitating a flexible and user-centric approach to testing.

We conducted a comprehensive series of manual tests covering every aspect outlined in our requirements document. Each test was carefully crafted to simulate user interactions and system behaviours, providing invaluable insights into the integrity and functionality of our software.

Our testing process involved executing a diverse array of scenarios, ranging from basic user interactions to complex system functionalities. Test cases were meticulously designed to cover all functional requirements, ensuring comprehensive validation across various aspects of the game:

- Game Start and Quit: We validated the system's ability to initiate and terminate the game from the menu screen, confirming that users could seamlessly navigate these fundamental functionalities.
- Character Interactions: We tested the user's ability to interact with the game environment, including character movement, collision detection, and interaction with map elements such as buildings and objects.
- Game Progression and End Conditions: We verified that the game progressed logically according to predefined conditions, such as character actions leading to game completion or termination, ensuring a coherent and engaging gameplay experience.
- Stats Tracking and Updates: We examined the system's capacity to accurately track and update player statistics in response to in-game actions, facilitating meaningful progression and feedback for users.
- Map Navigation and Transition: We assessed the functionality of map navigation and transition between different game environments, ensuring smooth transitions and coherent spatial navigation for users.

In conclusion, our hybrid testing approach, comprising a blend of automated unit tests and manual tests, was well-suited to the dynamic nature of our project. While automated unit tests provided rapid validation of code changes and ensured the stability of the codebase, manual tests offered a more comprehensive assessment of user experiences and functional requirements. Moving forward, continued vigilance in testing, coupled with a focus on addressing any identified failures, will be paramount in delivering a high-quality and user-centric product.