Software Project Plan

Jeremy Matthews

Colorado State University Global

CSC-480 Capstone: Computer Science

Shaher Daoud

Due Date: 05/25/2025

Software Project Plan

Testing is an essential phase in software development, helping to ensure the production of a reliable, performant software that meets user expectations and satisfaction. This report outlines a comprehensive testing strategy for *Ink Rollers*, my multiplayer Android game. The following sections detail the specific testing methodologies, critical use cases, inputs and expected outcomes, performance expectations, and anticipated failure conditions that are planned to be implemented into the project.

Testing Methodologies

Testing methodologies for this project consist of Unit Testing, Integration Testing, User Interface (UI)/ Instrumentation Testing, Performance Testing, and Failure/Negative Testing. These methodologies help to isolate issues and validate functionality systematically (Myers, Sandler, & Badgett, 2011).

Unit Testing

Unit tests will be conducted using JUnit and Robolectric, verifying isolated functions. For example, PlayerProfile.isValidColorSelection() will undergo testing to ensure unique color selections. MazeLevel complexity adjustments will also be tested to confirm cell counts match "LOW," "MEDIUM," and "HIGH" settings, ensuring maze generation integrity (Ammann & Offutt, 2016). Robolectric allows for testing Android components on the JVM without the need for an emulator, which makes it easier, faster, and more efficient to run unit tests (Vogella, 2017).

Integration Testing

Integration tests will use AndroidX's ActivityScenario and Firebase Realtime Database Emulator to simulate interactions between the system components. These tests will verify the seamless transitions between HomeActivity and MainActivity, ensuring that the Intent extras pass correctly, and validate the synchronization and data integrity with Firebase, particularly when hosting or joining games (Firebase, 2025).

UI/Instrumentation Testing

Automated UI tests will use Espresso and the Android testing framework to simulate user interactions. Specific scenarios include button taps for game initialization, joystick movements for avatar control, and mode toggling between “Paint” and “Fill.” HUD elements such as InkHudView, CoverageHudView, and TimerHudView will be verified through automated assertions to ensure accurate real-time rendering (Android Developers, 2025a).

Performance Testing

Performance tests will use Android Macrobenchmark and Microbenchmark libraries. Tests will measure consistent frame rates, targeting approximately 60 FPS under varying maze complexities and prolonged sessions. Memory usage, particularly concerning bitmap handling (PaintSurface), will be monitored to detect and prevent potential memory leaks. This will help to verify that memory growth stays within acceptable limits over extended gameplay sessions (Android Developers, 2025b).

Failure/Negative Testing

Negative testing will deliberately introduce faults, such as invalid game IDs, duplicate color selections, and simulated network disruptions using fault-injection methods. These scenarios will verify that error handling is robust, displaying appropriate user notifications, and that graceful degradation is supported (GeeksforGeeks, 2025).

Core Use Cases

Host a New Game

Tests will validate host game functionality by automating the selection of game duration (3, 5, or 7 minutes), maze complexity (Low, Medium, High), game mode (Coverage, Zones), and if the game is public or private. Tests will check if the settings are correctly stored in the Firebase Realtime Database and verify the waiting state until another player joins.

Join a Game by ID

Automated tests will input specific game IDs and validate that the settings retrieved from Firebase are accurate and that the waiting screen activates correctly until the host initiates the game.

Join a Random Game

Tests will automate joining attempts without specifying an ID, verifying correct handling through successful joins or appropriate error messaging when no sessions are available.

Gameplay Loop

Instrumentation tests will simulate precise joystick inputs, ensuring synchronized player movement across devices. HUD elements will be continuously monitored during gameplay to validate accurate updates on ink levels, coverage percentages, and timer countdowns.

Match End and Rematch Decision

Tests will confirm match completion logic by simulating scenarios where players select rematch options. Assertions will verify seamless restarts and state resets, ensuring correct navigation back to the home screen if a rematch is declined by either player.

Inputs and Expected Results

Scenario	**Example Input**	**Expected Result**

Host selects match settings	“Public”, “3 Minutes”, “Low Complexity”	Match is available for random joins and by Match ID, the match timer is set to 3 minutes, and the match maze generates with low complexity.
Join game with match ID	“IG7F26”	Match settings are retrieved from the real-time database, waiting screen is activated.
Joining game with no Game ID	No Game ID entered	Random match found or error message displayed if no matches are available.
Virtual joystick use	Drag joystick downward	Player character moves down on both player’s and opponent’s devices.
Ink refill toggle	Tap “Fill” mode button	Mode switches correctly, ink refills and is reflected in InkHudView.
Time expiration	Match timer reaches zero	Match ends, coverage is calculated, and dialog box displaying Win or Lose dialog box with option to play rematch.

Duplicate favorite colors in profile	Duplicate colors selected	Profile save blocked, error notification clearly displayed to user.

Performance Expectations

Performance tests are used to test for consistent frame rates around 60 FPS using controlled thread sleep intervals (Thread.sleep(16)). Another metric that should be tested for using performance tests is memory usage, crucial for PaintSurface, which should be optimized to prevent memory leaks and significant memory growth during prolonged gameplay sessions. Tests will target devices running Anroid SDK levels 26 through 34 to guarantee broad compatibility (Android Developers, 2025b)

Failure Conditions

There are several conditions that will result in failure conditions, such as entering invalid or non-existent Game IDs, network disruptions during Firebase synchronization processes, or attempting to save profiles with duplicate or invalid color selections. Tests will be conducted to simulated and validate these failure scenarios to ensure the game's ability to provide clear user notifications and perform graceful recovery and maintain stability and user confidence (GeeksforGeeks, 2025)

Conclusion

This detailed testing plan outlines the structured methodologies to ensure the robust functionality, seamless user interactions, optimal performance, and resilient error handling of the *Ink Rollers* game. Incorporating rigorous testing practices will significantly enhance user experience and software reliability.

References

Ammann, P., & Offutt, J. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press.

Android Developers. (2025a). *Espresso Testing*. Retrieved from https://developer.android.com/training/testing/espresso/

Android Developers. (2025b). *Write a Macrobenchmark*. Retrieved from https://developer.android.com/topic/performance/benchmarking/macrobenchmark-overview

Firebase. (2025). *Test your app with Firebase and Continuous Integration*. Retrieved from https://firebase.google.com/learn/pathways/firebase-continuous-integration

GeeksforGeeks. (2025). *Fault Injection Testing – Software Engineering*. Retrieved from https://www.geeksforgeeks.org/fault-injection-testing-software-engineering/

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). Wiley.

Sommerville, I. (2015). *Software engineering* (10th ed.). Pearson.

Vogella. (2017). *Using Robolectric for Android unit testing on the JVM*. Retrieved from https://www.vogella.com/tutorials/Robolectric/article.html