Project Proposal
Software Testing and Debugging
Group Members: Lidia Berhe, Maudeline Deus

Subject Software Application
Privacy Friendly 2048

Explanation of the Subject Application

1. What is the subject application?
    1.1. Privacy Friendly 2048 is a puzzle game where users are prompted with numbers on a grid and aim to add numbers of the same value together repeatedly (by swiping in a direction that would swipe the two numbers together) until either the number 2048 is reached, or if the user runs out of space on their board to generate new numbers.
    1.2. App Metrics
        1.2.1. Lines of Code: approximately 2800
        1.2.2. Number of Classes: 20 java classes composed of 12 activities including adapter classes, 3 database classes, with approximately 150 methods across all files. We will be focusing our testing on the GameActivity class since that is where the game logic is.
2. What is the application used for?
    2.1. Privacy Friendly 2048 is used to play the game 2048 (with full functionality) without sacrificing the user's privacy by forcing them to agree to any unnecessary permissions such as other 2048 applications.
3. What platforms is the subject application available on?
    3.1. Our subject application is available on Google Play, F-Droid
4. What set of users is the subject application aimed at?
    4.1. Privacy Friendly 2048 targets anyone who likes to play puzzle type games such as 2048 but do not want to give their own information for tracking mechanisms or advertisements.

Plans for Testing

Note that this application does not have a test suite already included, so we aim to produce as high a coverage as possible for a specific coverage criteria—specifically, we're aiming to improve branch coverage—by implementing our own test suite. In order to achieve our goal of improving the branch coverage of the test suite, we plan to conduct a variety of testing types, including:

1. Blackbox and whitebox
2. Mock testing
3. Property-Based Testing
4. UI Testing
5. IO Testing

Please note that some of these (excluding blackbox and whitebox testing) may change as we continue working on the tests in practice, as we figure out what is not feasible or relevant in relation to the project.

Test Tools and Frameworks

1. Espresso
2. Jetpack Framework
3. Mockito
4. jUnit5
5. Jqwik

Task Breakdown

To achieve our goal of improving the branch coverage of the test suite for the application, we plan to break down the different testing types and apply them to different parts of the project that each type is best suited for; we will be focusing our attention on the GameActivity file, as most of the game logic is found here. Below is our basic breakdown of what we plan to do, though tests may be added once we start working in order to either achieve our goal or holistically test the application. Additionally, while writing our tests, unless otherwise stated (such as in UI testing), we will aim to cover the source code and implement whitebox testing techniques.

Test Type	Tools	Task
Mock testing Tested by: Maudeline	Mockito jUnit	Create mock for Gamestate, check interactions in save, initializeState, updateGameState, readStateFromFile
		Create mock for gameStatistics, check interactions in initialize, readStateFromFile, saveStatisticsToFile
		Create mock for element, check interactions in initResources, initialize
		Create mock for Gestures, check interactions in setListener
Property-Based Testing Tested by: Lidia	Jqwik jUnit	Combining two of the same numbers is successful (is left with one number that is the sum of the two).
		Combining two distinct numbers is unsuccessful (the two numbers remain, and there is no combined number).
		Combining two distinct numbers is unsuccessful (the two numbers remain, and there is no combined number).
		When the highest number is 2048, then the user wins.
		When there are no more possible combinations, the highest number is 2048, and all the spots are occupied, then the user wins.
UI Testing Tested by: Lidia Note: While testing UI, we plan to	Espresso Jetpack	Clicking start new game starts new game on all of the different board sizes
		Clicking continue game reload previous game on all of the

implement blackbox testing techniques, such as, but not limited to, implementing a partitioning of input/output space into equivalence classes.		different board sizes
		Clicking the right arrow on home page changes board size to one size bigger or smaller a. Clicking too far in one direction stays on the biggest or smallest board size, depending on the direction
		Clicking menu opens menu navigation bar
		Clicking tutorial opens tutorial activity
		Swiping in a direction (right left, top, bottom) with a valid combination of two of the same number combines the numbers
		The interactions on the statistics plage a. For example, swiping between and clicking on the different pages
IO Testing Tested by: Maudeline Note: We will use integration testing will also be done here since these read and save methods depend on each other	jUnit	Test read, save game state a. saveStateToFile b. deleteStateFile c. readStateFromFile
		Create file to test gameStatistics a. saveStatisticsToFile b. readStatisticsFromFile

Notes

The following is an outline of what we found to be important classes and interactions of the application, for ease of understanding what, and how, to test items:

- GameActivity
    - setListener: handles swipes
        - onSwipeTop
        - onSwipeRight
        - onSwipeLeft
        - onSwipeBottom
    - updateHighestNumber
    - Check2048: winning stuff, message, start over
    - setDPositions
    - addNumber
    - gameOver
    - updateGameState
    - createNewGame
- Helper
    - GameState
    - GameStatistics
    - Element: box stuff
    - Gestures