Game Centre Walkthrough

CSC207H1 – Phase 2

Alex Quach, Jimmy Tan, Frederick Yao, Zuhab Wasim

Table of Contents

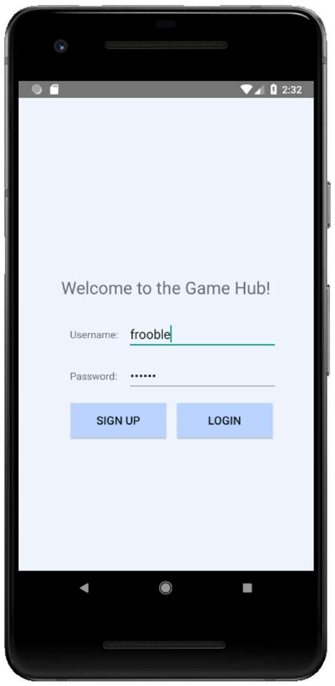
Front-End Tour

Launch Centre

The application starts with the user greeted with a login menu. The interface gives the user the option to create a new account using the **Sign-Up** button or log into an existing account by clicking the **Login** button with correct user credentials.

If the user attempts to create a new account with a username that already exists or attempts to log in with invalid incorrect login credentials, they will be unable to complete the process and will be provided with a message to explain their situation.

Additionally, the user may not create any accounts with incomplete fields hence each user must have a username, and password. Once the user has inputted correct log in credentials or a valid new account, they may enter the Game Hub.


Welcome to the Game Hub!

Username: frooble

Password: ••••••

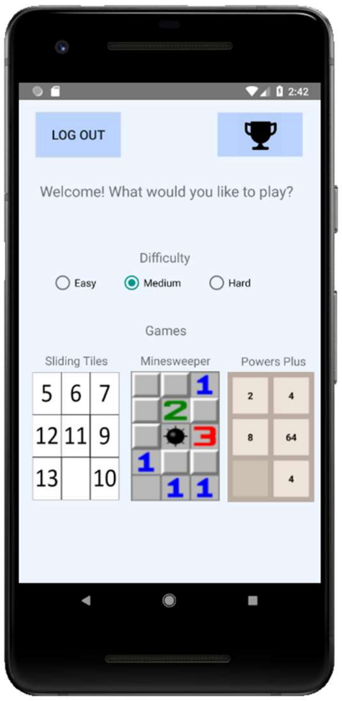
Game Hub

The user is then greeted with a Game Hub containing options for selecting the game they would like to play and on the difficulty they choose.

Initially, the difficulty of the game they would like to play is not selected and they must select one in order to play the game. The user can only select one difficulty at a time but can come back to switch it whenever they choose by returning to the game hub.

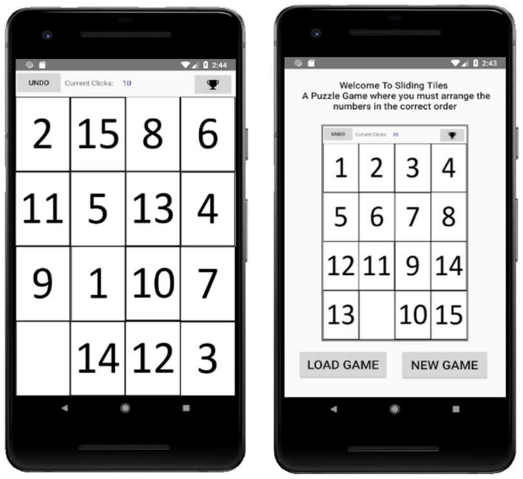
Near the upper corners of the interface contains a button with an icon of a trophy to represent the **leaderboard** button and one to log the user out of their account whenever they so choose. Logging out the user will not remember their previous log in information and the user must input it again to log in again.

Once the user presses any of the three buttons near the bottom of the interface, they will be taken to the chosen game with the difficulty they selected for an introduction of how to play.


Welcome! What would you like to play?

Difficulty: Easy [✓ Medium] Hard

Sliding Tiles



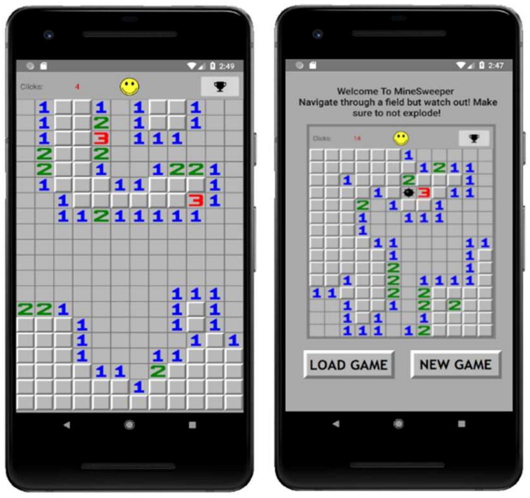
Welcome To Sliding Tiles
A Puzzle Game where you must arrange the numbers in the correct order

Our first game is an implementation of the Sliding Tiles game. The game is based around shifting tiles to sort them in ascending order from the top left, to the bottom right of the board given the restriction of shifting one tile at a time.

The user can click any tile the wish to select and if that tile is adjacent to the blank space, the tile will be shifted to that location. The difficulty of this game varies the number of tiles to sort given a 3x3, 4x4, and a 5x5 tiled board.

The user's number of clicks is displayed as the score they accumulate by each move made and if the user wishes to undo the move they made, they can click the **Undo** button near the top left of the interface. Additionally, they user can click the **leaderboard** button near the top right to see the scores at any time however, the users score will only be recorded if they have solved the puzzle.

Minesweeper



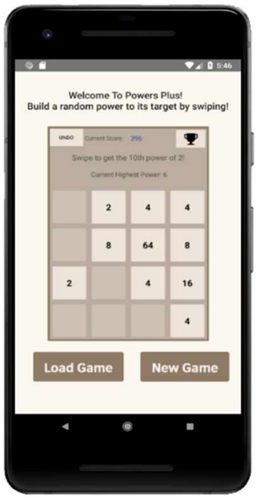
Welcome To MineSweeper
Navigate through a field but watch out! Make sure to not explode!

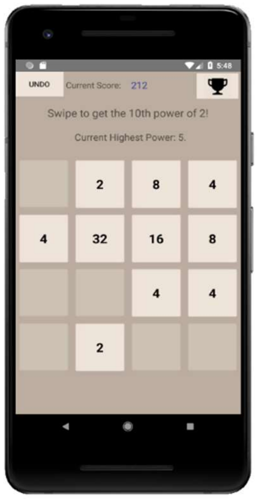
Our second game is an implementation of Minesweeper. The game requires traversing across a mine field and clearing out safe land while avoiding triggering mines.

The user can click on various cells in a board with dimensions depending on difficulty and reveal the specified tile. Clicking a tile excavates an area automatically but any tiles close a mine will warn the player by displaying a count of how many bombs are surrounding it. The user loses when a mine is activated by clicking on it.

Once again, the user's clicks are displayed near the top left of the interface to show the number of moves the user took to clear the mines. They have the ability to view the leaderboard by clicking the **leaderboard** button but will not have their score inputted until they have successfully cleared the board.

Power Plus



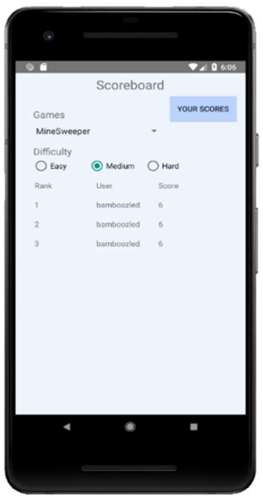


Our third and final game is a variation on the popular game 2048. The game consists of shifting the tiles on a board to merge and combine them to get to a certain target.

The user can swipe up, down, left, and right on a board to shift tiles and combine tiles of equal value. Combining tiles will increase the value by 1 power and once the target power is reached the user wins the game. If the user cannot make anymore moves, they have lost.

Each move that combines tiles together will increase the user's score by however many combinations they made with their values. The user can undo their move with the **undo** button if they so choose and can view the leaderboard at anytime. The game will only keep track of the score if the user wins the game.

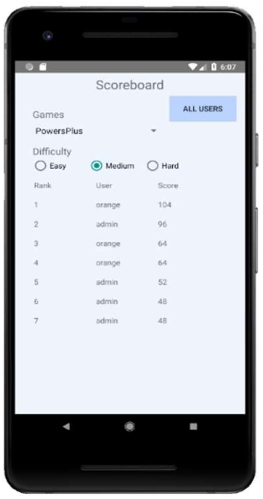
Leaderboard



Games: MineSweeper

Difficulty: ( ) Easy [✓ Medium] ( ) Hard

Rank	User	Score
1	bamboozled	6
2	bamboozled	6
3	bamboozled	6



Games: PowersPlus

Difficulty: ( ) Easy [✓ Medium] ( ) Hard

Rank	User	Score
1	orange	104
2	admin	96
3	orange	64
4	orange	64
5	admin	52
6	admin	48
7	admin	48

In the Game Hub, and amid playing a game, the user can choose to view the leaderboard to see the top 10 scores made by the other players who have played that game and difficulty.

The user can select the game they wish to see and must specify the difficulty of that game to view the scores. The board will display the rank, the username, and the score achieved of each respective user.

If the user wishes to see only their scores, they can click the **User Toggle** to switch between the top 10 scores of every user, or just theirs. The user's scores recorded are the 10 highest of the user and are recorded even if they do not qualify for the highest scores.

Back-End Code

Class Hierarchies, Super Classes and Abstraction

Tile

Tile.java, MineSweeperTile.java, SlidingTilesTile.java

These files represent a touchable/clickable tile for a game that requires the user to simply tap a tile to make a move. MineSweeperTile and SlidingTilesTile extend the superclass Tile in order to avoid duplicate code and force these two tile classes to override essential methods that are needed for this type of Tile. PowersPlusTile does not extend Tile because it is a sliding game, rather than a touching game.

StartingActivity

StartingActivity.java, MineSweeperStartingActivity.java, SlidingTilesStartingActivity.java, PowersPlusStartingActivity.java

Since it was noticed that all three game starting activities had the exact same method names and field names, and future games might also have the same issue, it was decided that a superclass for the StartingActivities that extended AppCompatActivity would be created in order to avoid all the duplicate code and to force inheriting classes to implement the necessary methods that all Starting Activites will have (i.e, addLoadButtonListener. addStartButtonListener, switchToGame)

GridViewAdapter

GridViewAdapter.java, TileAdapter.java, LeaderboardAdapter.java, PowersPlusAdapter.java

These files represent activities that need a special tile for the display GridView. Any GridView that needs a special tile for display will extend GridViewAdapter so they avoid duplicate code and inherit the correct methods and fields needed to be a proper grid view adapter. This is needed to properly display tile like objects onto a GestureDetectGridView

GestureDetectGridView

GestureDetectGridView.java, MineSweeperGestureDetectGridView.java, SlidingTilesGestureDetectGridView.java

This superclass was created based off of someone else's code in order to create a GridView that was able to detect a Gesture in order to make a move. Thus, games that require a touch move like MineSweeper and SlidingTiles had to extend this grid view in order for the game to function properly.

User Interfaces

Launch Centre

LaunchCentre.java, activity_gamelauncher.xml

These files represent the activity (.xml) and functioning code (.java) for the Launch Centre. LaunchCentre.java includes button listeners for **Sign-Up**, and **Login**, that manages users entering the Game Hub.

Since this utilizes users, the class instantiates a user manager object that deals with the logic relating to creating and retrieving users. Additionally, a file manager object is also instantiated to deal with saving and loading users within a local serialized database.

Finally, the launch centre will start the game hub activity when logging in is successful.

Game Hub

GameHubActivity.java, activity_gamehub.xml

The game hub activity acts as the programming for the respective activity. It includes listeners for logging out by reverting the activity to launch centre, starting the leaderboard activity, changing the difficulty, and executing the various games.

In this class, we instantiate a file manager object of type scoreboard manager to manage the scores we will retrieve or save for each game. scoreboard manager is serializable, and we will save all scores to a locally serialized database.

The class also retrieves the difficulty input from the user and passes it to the next game the user will play. For our purposes we have defined difficulty as set to “Easy”, “Medium”, and “Hard”, that will remain the same for any game.

The game hub will start the starting activities for each game, on any game button click.

Leaderboard

LeaderboardActivity.java, LeaderboardAdapter.java, activity_leaderboard.xml

The leaderboard activity acts as the back-end programming for the respective activity. The class includes listeners for changing difficulty via radio buttons, changing the game selected via spinner, displaying via grid view, and changing user-only or all-user scores.

In this class we instantiate a file manager object of type scoreboard manager to manage the scores that will be displayed onto the grid view. Since the leaderboard is only used to display the scores, the scoreboard manager will be loaded and never saved to.

The class holds constants that track the file names for each game and a default file to load as well. Once input is received from the user (or obtained by default), the grid view is displayed to using the scoreboard manager by setting the contents of a list of scores to the leaderboard adapter.

Starting Activities

SlidingTilesStartingActivity.java, MineSweeperStartingActivity.java, PowersplusStartingActivity.java

These files are used to program the initial screen of each game where the user can save and load the game. Each file instantiates a file manager for types user manager, scoreboard manager, each game's board manager. These allow the load and saving of the user, user's scores, and the user's game as all these types are serializable.

This class will instantiate and saves a specified games board manager to a temporary location to be used when the game begins. It also holds the button listeners to create a new game, or load game if there exists a save.

On completion, this class will start the respective game's game activity.

Sliding Tiles

SlidingTilesBoard.java, SlidingTilesBoardManager.java, SildingTilesGameActivity.java, SlidingTilesGestureDetectGridView.java, SlidingTilesMovementController.java, SlidingTilesTile.java, slidingtiles_main.xml

These files are responsible for running the Sliding Tiles game. Every time the board gets updated, since it extends observable, it notifies its observer (sliding tiles game activity) to display the results onto the screen. Any input is received from the gesture detect grid view that returns a position where the user has tapped on the grid view. Each move is made by swapping a blank space with the clicked tile if the clicked tile is validly adjacent.

The sliding tiles board manager updates and manipulates the sliding tiles board and score of the game. The board consists of manipulating the sliding tiles tile on the board per the user's request. When the game is over, the game activity disables the user input until a new game has started.

When a user wishes to undo a move, the board manager will reload a previous state of the board using a state manager.

Minesweeper

MinesweeperAdjacencyCheck.*java*, MineSweeperBoard.*java*, MineSweeperBoardManager.*java*, MineSweeperGameActivity.*java*, MineSweeperGestureDetectGridView.*java*, MineSweeperMovementController.*java*, MineSweeperTile.*java*, minesweeper_main.xml

These files manage and control the Minesweeper game as well as updates the activity on updates to the minesweeper board since the board is observable. All input on the board is recorded by the gesture detect grid view for a position of the cell and updating, and movement controller to detect taps.

Each tile clicked by the user is taken from gesture detection and inputted into the board manager to make moves and updating score. The board manager manipulates the board and the board manipulates each tile.

Each move results in the game being lost if it is a bomb, the number underneath, or recursively flooding out a cleared area of no bombs until numbered tiles are found. Once the game is over, the gesture detection is disabled until a new game is made.

Powers Plus

PowersPlusAdapter.*java*, PowersPlusBoard.*java*, PowersPlusBoardManager.*java*, PowersPlusGameActivity.*java*, PowersPlusOnSwipeListener.*java*, PowersPlusTile.*java, powersplus_main.xml*

These files manipulate and control the Powers Plus game the user is played. Each move is made from detection of a swipe using an on-swipe listener object that detects if a gesture for swiping up, down, left, or right is made.

On a valid gesture, the board manager updates a board by shifting and merging tiles appropriately. Since the board extends observable, it notifies the game activity to display the results of the move. That is, the new score, board, and highest power.

The board manipulates and merges tiles when necessary and if the user wants to undo a move, a previous state of the board is displayed by using a state manager, and also the previous score the user had.

If there are no moves or the user has won the game, the swiping is disabled since the game is over until the user creates a new game.

Data Managers

File Manager

FileManager.java

This class is responsible loading and saving instances of the class saved to the local serialized database within the app that implement serializable. This class can be of any generic type T and is used for any file that will be saved or loaded during the execution or return of this application.

Saved Games

SavedGamesManager.java

This class holds user's easy, medium, and hard save of any game. This class is generic and can work on any type T to be saved and we utilize it for saving a user's arbitrary game's saves for future use. This class simply loads and saves the desired difficulty saves needed.

Scoring

Score.java, ScoreboardManager.java

This class holds three different scores for the user of easy, medium, and hard. This class although not generic is used to represent every game's score manager. Each user will have a scoreboard manager relating to their scores and will be saved locally due to this class being serializable.

It contains the methods to return a list of scores by difficulty, user, and game and it is primarily used by the leaderboard for display but is saved throughout all games.

State Saving

StateManager.java

Each user that is playing a game will have multiple states of a game they are playing from making moves. The state manager allows for saving of these states in similar fashion to a stack.

It contains method to save a current state to the state manager and to remove the most recent state saved to undo the move the player made only if the state manager is not empty.

User Tracking

User.java, UserManager.java

Each users account information whenever used is recorded in the user manager that manipulates and contains multiple users. Each user harbours a name, password, and saves of the games they have played.

Since the user manager is serializable, all users are saved locally in the phone for later use. Each user is loaded from the user manager in almost every activity of the game for the requirement of all personal properties of the user, the games and scoring tracked.

Unit Tests

Our unit tests have achieved a value around 93% coverage of our testable code. To get this we had excluded files that had been deemed untestable or not required to test:

Untestable Code

AndroidManifest.xml, res, generatedJava,

These files were considered to be untestable. The android manifest file has no logic and only sets properties thus there is nothing to test. The res package holds only image and mark files that once again have no logic to test for, and lastly the generated Java only exists as the compiled data of our application and thus not our direct code to be testing.

Adapters, Models, and Views

LeadboardAdapter.java, GridViewAdapter.java, GestureDetectGridView.java, SlidingTilesGestureDetectGridView.java, SlidingTilesMovementController.java, PowerPlusOnSwipeListener.java, PowersPlusAdapter.java, MineSweeperAdjacencyCheck.java, MineSweeperGestureDetectGrideView.java, MineSweeperMovementController.java

These files exist only to detect input from the user or to display back to the activities they relate to using adapters and grid view controllers. Since the only logic in these files pertain to getting and displaying interactions, this code should in theory work by default as all it does is call other testable classes' methods to do whatever it needs (e.g saving highscores). Moreover, the only true way to test this code is to simulate the application screen and touching itself which cannot be done using Junit (easily). Thus, we will exempt these files from testing.

Starting Activities

MineSweeperStartingActivity.java, PowersPlusStartingActivity.java, SlidingTilesStartingActivity.java

Starting activities set up the initial interface before the game is to actually run. The code in these files only manipulate the buttons on their interface and shift to the relative main interfaces of each game. Once again there is no code that can be tested for logic, these files are exempt as well.

Game Activities

MineSweeperGameActivity.java, PowersPlusGameActivity.java, SlidingTilesGameActivity.java

GameActivities display the actual games using the appropriate classes. Since the code in these files only manipulate the GridView and other interface items, and have little to no logic, it is untestable using Junit tests. We also ensured that the only thing these GameActivities do when touching other testable classes is call those testable classes' methods so there is no actual logic contained in the methods of these GameActivities. Also, if we were to create a controller for this activity, the code inside the new controller has already been tested in their main class's JUNIT test files. Thus, we excluded all GameActivities from testing.

Some Design Patterns

Iterator

MineSweeperBoard.java, PowersPlusBoard.java, SlidingTilesBoard.java

We implemented our own implementations of Iterator for each game's board. These iterators helped solve problems relating to 2-dimensional loops for rows and columns, by flattening many lists and access them linearly by getting the next element of the board. This made the code cleaner and easier to understand.

Grid View Adapter

TilesAdapter.java, GridViewAdapter.java, LeaderboardAdapter.java, PowersPlusAdapter.java

The grid view adapters we use allow for simple display onto grid views on any game's activity. We implemented our own generalized, and specific adapters for each game that when required to update the board, simply are a parameter to pass into our grid view controls for display.

Serializable files

FileManager.java, User.java, UserManager.java, All BoardManager.java, All Board.java, Score.java, ScoreboardManager.java

To save games, load games, keep track of high scores, and store users in our user database, we used serializable files and implemented a FileManager.java that took care of loading and saving files in all the activities. The FileManager was implemented as a generic class to ensure any object that implements serializable could be saved into a file on the phone. These helped solve the problem of keeping track of who signed up, all the saved games, all the highscores for the scoreboard, without the use of a database and database language.

Gesture Detector

PowersPlusOnSwipeListener.java, PowersPlusGameActivity.java

The gesture detector powers plus implements the on-touch listener with views. The inclusion of this design pattern allowed for swiping to be the means of controlling the game in a clean and responsive way.

Stack

StateManager.java

Through our implementations of the undo functionality for sliding tiles and powers plus, we noticed that recording each state of the game is accessed as a stack. We implemented our own stack to be utilized for the states of each user's games that makes the code appear cleaner and leaves the need of unnecessary methods from Java's implementation of the Stack that can take valuable space in the application.

Observable

MineSweeperBoard.java, PowersPlusBoard.java, SlidingTilesBoard.java, MineSweeperGameActivity.java, PowersPlusGameActivity.java, SlidingTilesGameActivity.java

By implementing observable, our classes had access to any change made by other classes easily without creating fields for both files. With observer, any updates on a games board notifies the game activities to update the display in a clean fashion.