CSC207 Group 0614 Phase 2 Walkthrough

Welcome to our game center for CSC207 fall 2018.
We are Lizard King Software LTD aka group 0614.

Introduction and Login Functionality

To use our game center, first you must register an account (demonstrate)

- After you fill the form and click register, the app will communicate with our server database through the link https://kayyzz.com/Register.php using the StringRequest class from the android.volley package.
- This Register.php code will direct the request to the database and and store your username, name, email, and password on a database table using SQL.

Logging in does the same thing but in reverse.

- After you click the Login button, LoginRequest.java sends the inputted username and password to the database through https://kayyzz.com/Login.php and SQL and checks to see if the username and password exists. If it exists, a success message is sent back and lets you into your game centre account.

Quick Game Class Demonstration

Quick set-up and run through of most games with a brief touch on relevant functions unique to the game

- Undo, save game, and interface interaction will be shown

Key Classes

GameManger.java

- Master class for control over each game's specific functionality and implementation

Scoreboard.java

- Separate entity to allow tracking, calculation, and display of scores per game

GameActivity.java (among each of the three games)

- Controls UI for each of the games, also helped drive class design within each game

Adaptors (among each of the three games)

- Allowed for updating the UI, vital to functionality as an app

Utilized Design Patterns

Obersable, observer

- Cleaner communication between different components of the app, namely between UI and actual game code

Factory Method

- Used to produce board and tiles in SlidingTiles
- Made testing easier
- Cut down on required use of static variables during game setup

Serializable

- Used for game saving and game loading functionality

VMC (view model controller)

- Better testing, as models and views do not have to be tested
- Keeping functionality separate allows for better troubleshooting
- Great design for an app in which each game has largely same structure in terms of how its UI and game code work together

Iterator

- For loops are honestly pretty great for a lot of functions within games, especially grid based games where sometimes lists of tiles need to be looped through
- Iterator makes that much easier for us to do

Scoreboard Design

Scoreboard is an abstract class with several instance variables to account for the level of complexity of a game, the duration that a game play lasts, and the number of moves and undos the current player of a game has made. Each game will have a child class that extends the scoreboard class and implements the abstract method calculateScore() in a way that is meaningful to the individual games themselves. This method also stores the newest score in two sorted arrays that are instance variables of the child class that belongs to the game intended for tracking the global high scores and the user's high scores, respectively, so that when a game ends, a text bubble displays the new score, the user's highest score, and the game highest score. Every new score, along with the corresponding game name and the current user's username,

is written into a save file upon creation for the score page to display using the methods in the “[Game Name] ScoreBoard Activity” class in each game.

- How to interpret the scores: In Sliding Tiles, the score is calculated only when a game is won, and naturally the user who plays the hardest version of the game with the largest board in the shortest amount of time earns the highest possible score. In MineSweeper and 2048, the scores are updated constantly in the background during game play after every click or slide movement, and the calculateScore() method is called when the player either loses or wins the game. In MineSweeper, the more of the board is revealed at the end of the game, the higher the score will be, and the amount of time that it takes the player to achieve that score will negatively affect the score but not to a significant extent. In 2048, the scoring system precisely follows the original game’s scoring system: every merge between two identical blocks creates a sum that is added to the score.

Unit Test Coverage

Minesweeper (boardmanager and tile classes) have 100% test coverage. Most other classes save for activity, UI, and view related classes are covered to varying degrees. The classes not covered are due to the fact that unittests are either impossible to for them or because the functionality which can be covered by unittest is minimal/ largely irrelevant to test.

Conclusion

Hopefully you understood and can identify the same reasons we did behind our design choices when creating these two new games. *Answer any questions which might be remaining*

If there is enough additional (likely not the case), walkthrough some more of the cooler additional functionality such as URL image uploading in SlidingTiles.