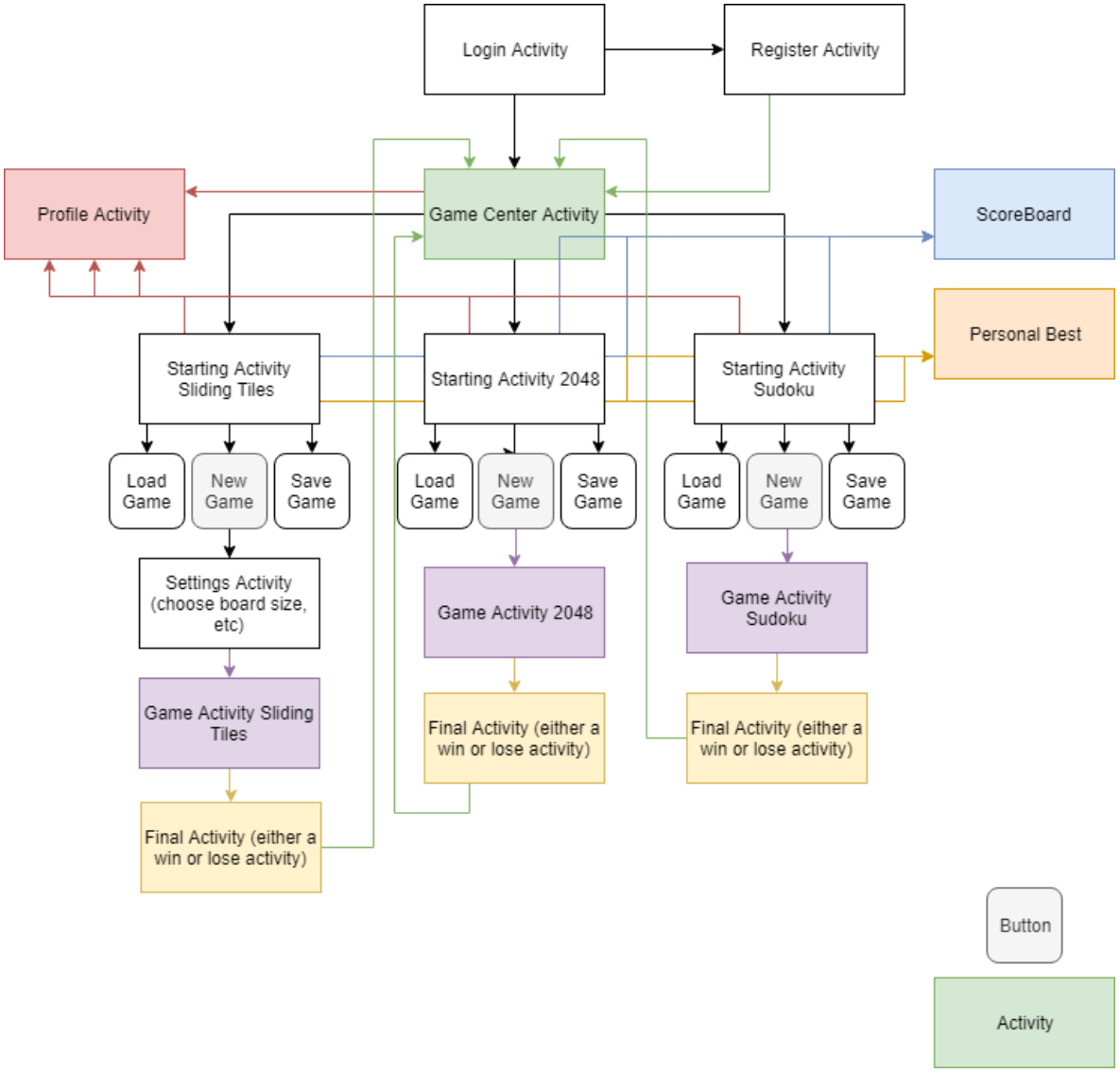
CSC207 Software Walkthrough

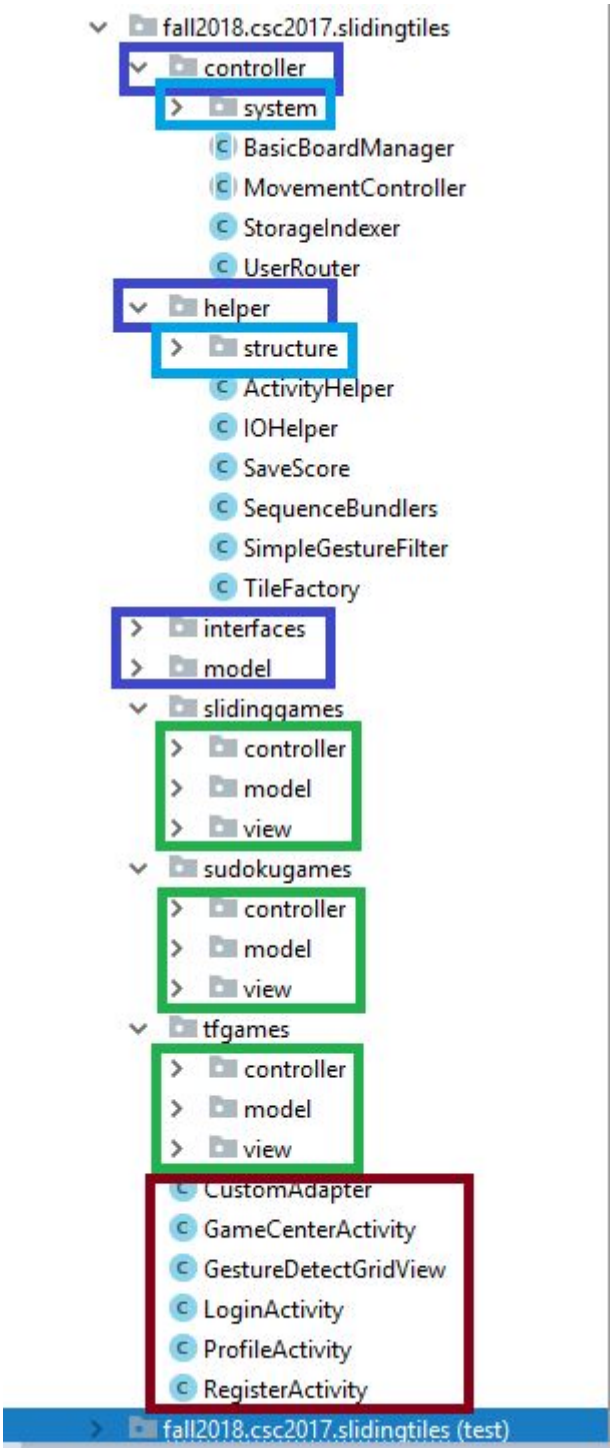
Group 0618: Team Memebers and Responsibilites

1. Timothy Lee
    a. Responsible for front-end design (UX/UI) [color scheme, visualization]
    b. Primarily worked on the majority of all xml's (layout and drawables), Activity classes, powerpoint/walkthrough.pdf and help with some helper functions/refactoring process
        i. implemented gesture detection (for swiping gestures in 2048 game)
        ii. implemented inputminmaxfilter (to specify and force limit the maximum number of undo steps in the range of 3 to 100000).
2. Junxuan Wu
    a. Designed the verification system, scoreboard system and some ui structure design. ActivityHelper, IOHelper, sequenceBundler, User, UserRouter, LoginActivity, TileSettingsActivity, RegisterActivity, ScoreBoardActivity, PersonalScoreBoard Activity. Do some of the refactoring part(mostly with Yinling Luo). Find and fix most bugs of code.
3. Quanzhou Li
    a. Implement undo function
    b. Implement a stack to store each move the user makes in a game, enabling the user to undo by popping from the stack. The maximum number of undo steps is changeable (Before a game starts).
4. Yinling Luo
    a. Implement complexity function: Make Tiles callable for boards of all complexity, and generate corresponding images for 5X5 game.
    b. Implementing the complexity part in tileSettings and initialize the game activity.

Please see TEAM.md for more details

Below is a diagram of how our “Activity” classes interact with each other from the user’s perspective (i.e. each activity is what the user will visualize while using our app):





Important Classes and their Implementation:

Firstly, as per the screenshot of our package organization on the right, we have tried to organize each individual class into its corresponding location in as much detail as possible to ease the marking process but also for intuitive purposes. For example, under the package “helper”, each ‘helper class’ is often re-initialized several times in other classes to perform the same, repeated operations, which help avoid duplicate code and large classes/methods.

Now, we will use some of our CRC cards to demonstrate a few selected examples that are crucial to our program:

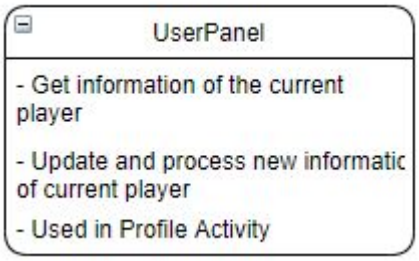
1. Game Cache System (under package ‘system’)

GameCacheSystem
- Load Game
- Save Game
- Update Game State
- Retrieve Data
- Get previous game

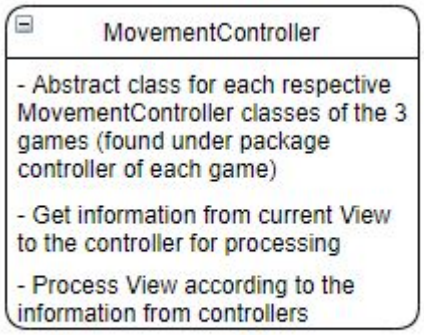
2. Score Board System (under package ‘system’)

ScoreBoardSystem
- Load Score
- Calculate Score
- Sort Score (ascending order)
- Provide algorithm and interact with the 3 games' score activities (SlidingScore, TFScore, SudokuScor under the package 'model' for each respective game

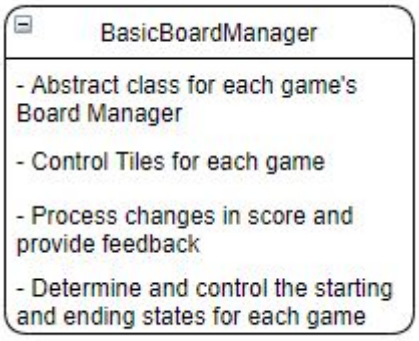
3. UserPanel (under package ‘system’) [Bonus for extra feature in Phase 1]



4. Movement Controller (under package ‘controller’)



5. BasicBoardManager (under package ‘controller’)



Unit Test Coverage: An Example

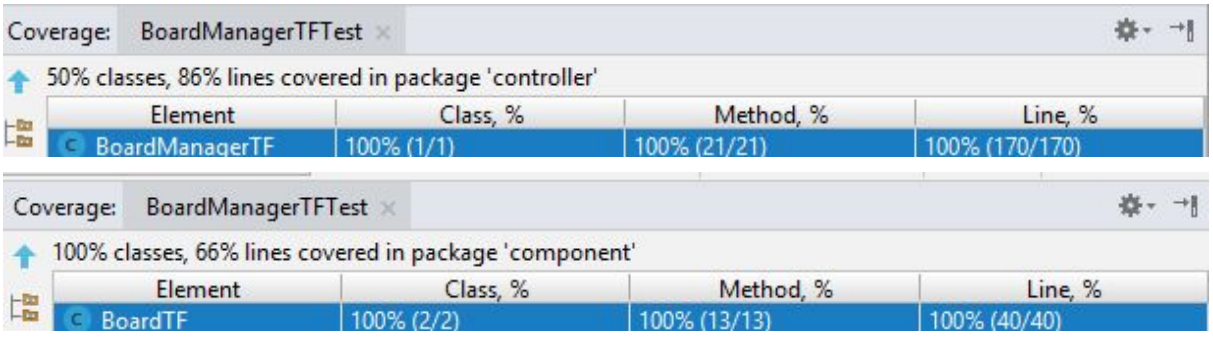
We have written a complete unit test class (under 'test' package) with 100% coverage for

- 2048 Board Manager and 2048 Board (see below)

We have also completed test files for

- Sudoku Board and Board Manager
- Sliding Tiles Board and Board Manager

Throughout the project, we have attempted to proceed with Test Driven Development (TDD) when designing the classes in the first place-- that is, we first wrote the tests for functionality testing then proceeded with writing the method itself. Unit tests are only written for controller and model classes (view/activity classes at the UI level).



Design Patterns and Why we chose them:

Factory Design Pattern:

- since all of our games are tile-based, the factory design pattern enables us to simply call methods such as <createTile()> and pass in the type wanted and corresponding values.

MVC:

- Following Model–view–controller pattern, we have sorted each class to these specified packages for each of the three games.
- For example,
  - Model (tiles, board and scores);
  - View (activities and action listeners);
  - Controller (boardManagers and Movement Controllers).

Singleton:

- An example would be a GameCacheSystem, where we create a global system for one user at a given time.

- This avoids shotgun surgery, since we just have to change or update the code in the singleton class rather than all 3 games as all 3 games have the same behavioral properties.

Scoreboard: Implementation and Intuition Behind

Front-End:

- For the visual design of the Sliding Tiles scoreboard system, we have chosen to implement a local version of the scoreboard for the current user which could be accessed by clicking the button “Personal Best”, and also a global scoreboard, which takes into account the ranking for all registered users. Both buttons could be accessed in the starting activity.
- For simplicity and clarity, we have chosen to simply include the top 3 players for both local and global scoreboard of Sliding Tiles game including all 3 different board-sizes. For the other two games, we have only included the top 5 global players in the scoreboard.
- The score for each game is calculated as follows:
    - Sliding Tiles: least number of moves (touches) to solve the board
    - 2048: least number of moves to get to the 2048 tile
        - Example for 2048:

- Each game’s scoreboard is also inherited by the super class “BasicScoreBoardActivity”, which provides the primary layout for displaying the top high scores in the format “username”, “score” (sorted in least number of moves taken)

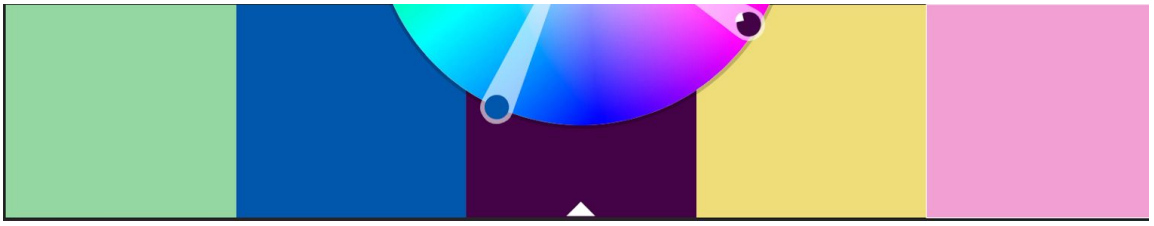
Back-end:

- The scores (number of moves) will first be displayed whenever a game is completed (as per the figure above) by each game’s corresponding activity and layout class (i.e.

"activity_final_score_tf.xml" under layout package and "FinalScoreTFActivity" under 2048's view package) when a game is completed (either a win or lose).
    - We have first created a "BasicFinalScoreActivity" that is used as a father template to be extended from each game's score activity.
    - In this super class, we will be able to get the current board manager for each individual game by accessing the current user through class "UserPanel".
- In each game's respective (global) Scoreboard Activity:
    - The high scores are managed and stored by a hashmap with <key: username, value: array of scores>. Each game has each own corresponding file for its respective username and all of his/her scores.
    - Each game's scoreboard activity will also initialize Scoreboard System (under package 'system'), which will help determine and sort the high scores (and their corresponding users) in the specified format.

A little about design:

- For aesthetics and consistency purposes, the entire design of the program is revolved around the central color scheme as follows (selected from Adobe Color CC):



    - The color scheme is not only applied to font colors, but also to backgrounds and images/drawables as well (using the "match color" tool from photoshop to match the original image color to this color scheme)
- For a more intuitive visualization, we have tried to incorporate the design concept of skeuomorphism as much as we can, which could be found throughout the entire program such as our Image Buttons (i.e. undo and profile button). A demonstrative example would be the game center, where each game is represented by self-explanatory Image Buttons to avoid as much excessive usage of text as possible and to achieve a more user-friendly and intuitive design.