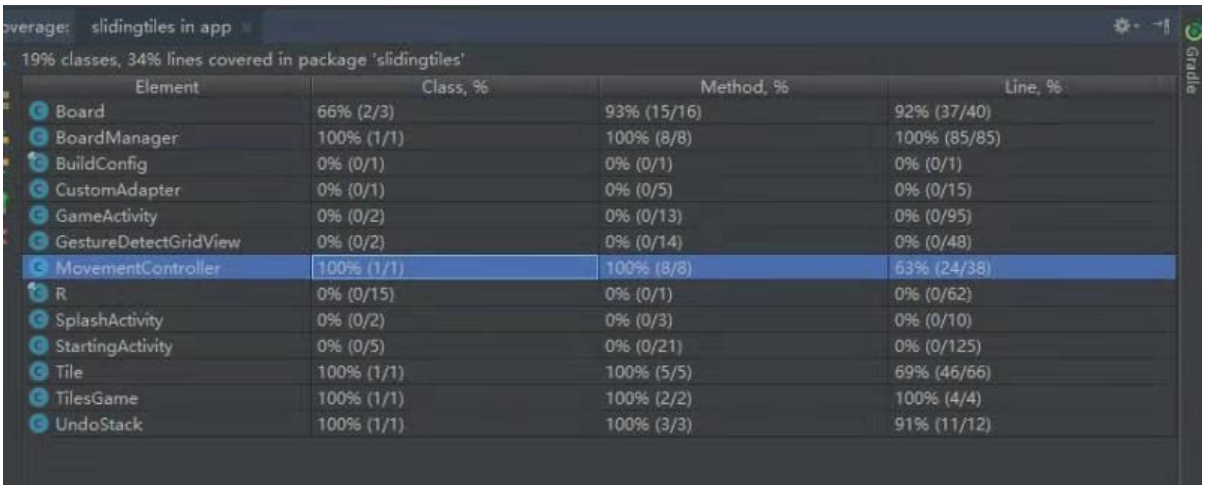
- **What is your unit test coverage?**

Sliding Tile Game Test Coverage:



Board uncover parts: Iterator method and class, toString method can not be tested (toString is a override method).

MovementController uncover parts: There are 4 others if statements remain untested.

Tile uncover parts: There are 25 switch cases, and only 2 of them were tested.

Undo Stack uncover parts: The attribute for changing the size of the stack was not tested, since we hadn't use it functionality.

All the remianing activity classes: We tested ui manually.

Connect4 Game Test Coverage:

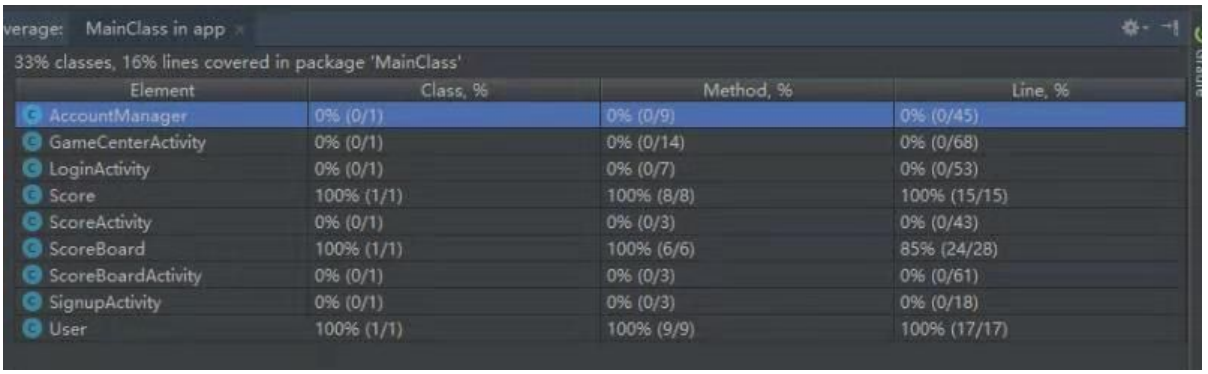
Activities uncovered part: Activity can not be tested. We test manually.

Grid Generator: Adaptor can not be tested.

Mine Sweeper Test Coverage:

Activity classes covers 0%, since it can't be tested. But we tested it manually.

Main Class Test Coverage:



Since we implement singleton on AccountManager, it can't be tested.

And another four classes are all activity classes which can not be tested.

- **What are the most important classes in your program?**

MainClass package: AccountManager, User, ScoreBoard, Score

Connect4game package: GridManager, ComputerAI

SlidingTiles package: Board Manager

MineSweeper pagekage: MineGameActivity, MineGame, MineManager

- **What design patterns did you use? What problems do each of them solve?**

List of design patterns we used:

Singleton:

We implement Singleton on AccountManager, which is responsible for user sign up and login. The save and load methods in AccountManager allow us to reused

the data even when the program terminates. With the help of this design pattern, we don't have to pass this instance over activities and save it to specific activity.

Factory:

We implement Factory on Scoreboard, which is responsible for saving all the scores for different players in different games. When getting scores for one player for one game, we use the factory to generate different score lists for different game for different players as well as the score lists for different game for all players.

- **How did you design your scoreboard? Where are high scores stored? How do they get displayed?**

Our scoreboard is designed using a factory. We created a class called Score, which takes in an int representing the score an user got, a string representing the user's name, and a string of game name. Using the class Score, we can easily get desired scores of different games or users.

After each game is finished, a Score object containing the score of current game, name of current player and game will be stored inside ScoreBoard class, for future use of generating required scoreboards.

There are two methods in ScoreBoard: getGameScoreBoard and getUserScoreBoard, both specified to generate desired scoreboards. The getGameScoreBoard method will generate a ScoreBoard of certain game.

To get a scoreboard of certain game, we first look at every single Score stored in ScoreBoard, and compare the name of game we wanted with the game name stored in each Score. Then an ArrayList containing all Scores of the same game will be returned, and then sorted by the sortScore method. The scoreboard for certain user is attained similarly.

After we get a list of wanted Scores, they will be displayed on the Scoreboard page. There are two pages: Your Scores and Top 5s. In Your Scores page, the top three scores of each game will be displayed, and in Top 5s page, top 5 of each game will be displayed.