- Code Coverage:
  39% line coverage
  41% method coverage.
- Most important classes in the program:
    1. UserAccManager. The user account manager is one of the most important classes in our Game Center, because it manipulates the data that are stored in UserAccounts, which includes, the username and password, game saves for each user depending on different games, as well as the number of undos each user chooses to have. This class provides methods that can access and update those information.

    2. Merge2048. This class is the most important one for game 2048. It is responsible for combining elements in rows/columns in the board, which is the fundamental feature of the game 2048. It combines two adjacent tiles to generate a new tile if they share same number, and skip all blank tiles.(Method class)

    3. ImageOperation. This class contains all the methods needed to set up the background image for each button, which includes resizing image, cropping image, and superposing images. It acts as a factory that produces Bitmap, which is the main object that we use to accomplish various Phase 1 image bonuses as well as the create and update button methods in different games.

    4. FileSaver: The utility class that is used to save/load object from files. It has 2 public static methods. Save to file, which basically open a file output stream and put the object in the parameter into file location given in the parameter.
    Load from file, which opens a file input stream, and load the object from file in file location given in the parameter.

- **Design Pattern:**
    - **MVC:** All Three games have applied MVC design pattern. GestureDetectGridView class corresponds to view, which is the board that the user can see. BoardManager is the model class, which stores data like board and manipulate on the board. MovementController is the controller, which is responsible for processing movement like sliding or tapping.
    - **Iterator:** Game Sudoku is the typical game that used this design pattern. We have three types of iterator in SudokuBoard which are SudokuIterator, SudokuVerticalIterator and SudokuSectionalIterator. Those iterators correspond to iterator in columns, rows and single tiles (Since when we tap the tiles, it will one by one follow from 1 to 9) .
    - **Observer:** Each Board class extends Observable and its corresponding GameActivity class implements Observer. Therefore, whenever a change on a Board is been made, its GameActivity will notify the change immediately. This also contributes to the View part of MVC.
    - **Strategy:** Since the user account manager is in charge of adding score to each player, the add score function is defined in the UserAccManager class and it need to have different adding score algorithms deciding on run-time,

depending on which game user's currently playing on. Therefore I use the strategy design pattern, created 1 ScoringStrategy interface, with 3 seperate strategies for 3 games implementing that interface, and in the adding score method, we pass the strategy in as a parameter, and it's type is the general interface ScoringStrategy, when different games called for add score, they can pass in different strategies to calculate score, thus enabling the ability of choosing algorithms on run-time.

- **Scores and Scoreboard**
    - Our scoreboard is designed using a page viewer. Page viewer is an Android Studio design that allows the application to display different pages of informations within the same Activity. When the user enters the scoreboard activity, he/she can slide right or left to view the scoreboards for all the games. An user account will have a Map that stores his/her highest scores of all his/her games, and the UserAccManager will have informations for all users. When the user launches to the ScoreboardPublicActivity(which is our scoreboard), this activity will display different pages of ScoreAdapter. The ScoreAdapter will use Score.class to generate the rank of the username and their scores. This rank will be displayed on the page of a specific game.
- **Undo**
    - We implemented Stack from java.util in order to implement the undo function. HistoryStack saves some information (depends on games) from every steps of board in BoardManager. Since BoardManager is saved, HistoryStack is also saved. When users undo a step, we only need to pop the last score and list of tiles from ScoreStack/HistoryStack and set the score and board.