Sequencer:

Sequencer is a game where the goal is to remember a sequence which is 100 items long. The first round of the game requires you to remember the first item, the second round, requires you to remember the first 2, the third requires you to remember the first 3 and so on and so forth. This way, The first few round of the game are quite easy. But as you have to remember more and more items in the sequence, the game becomes harder.

Tic Tac Toe Features:

Tic tac toe is one of the other two games we decided to implement. There are game modes to choose from in Tic Tac toe; player vs player, player vs random AI, and player vs a somewhat strategic AI. In this game, the user always plays first, and marks their circle with a red marker, and player 2 with a blue marker. There is a countDown timer that allows for us to keep score in the leaderboard. The score is calculated and kept by the shortest time it takes to beat the AI. There is no score kept when playing player vs player mode. The timer starts at 100 seconds, and whoever wins first, ties or if the timer runs out decides how the game ends.

- What is your unit test coverage?

  Our unit tests cover any controller code. This includes methods from these classes

  Sequencer

  - SequencerBoardManager
  - Sequence

  Sliding Tiles:

  - SlidingTIlesBoardManager
  - SlidingTIlesBoard
  - MoveStack
  - SlidingTIlesTile

  TicTacToe:

  - TicTacBoard
  - TicTacBoardManager
  - TicTacMinimaxStrategy
  - TicTacMarker
  - TIcTacRandomStrategy

  ScoreBoard:

  - LeaderBoardController

- LeaderBoardReader
- Scores
- UserScores

This total code coverage excluding the model and view classes is :

The classes excluded from testing are all model and view classes:

These include classes from :

Sequencer:

- SequencerGameActivity
- SequencerMovementController
- SequencerGestureDetetctView
- SequencerStartingActivity

Sliding Tiles:

- CustomAdapter
- SlidingTilesGameActivity
- GestureDetectGridView
- MoveStack
- SettingsActivity
- SlidingMovementController
- StartingActivity

TicTacToe:

- TicTacCustomAdapter
- EmptyStrategy
- TicTacGameActivity
- TicTacGridView
- TicTacStrategy and TicTacEmptyStrategy
- Timer

ScoreBoard:

- LeaderBoardCustomListAdapter
- LeaderBoardFrontEnd
- swipeTest
- swipeViewAdapter
- DemoFragment

Further we do not test GameActivityModel since it requires mocking the tests,

In addition, login, registration, and starting activities are not tested since the code in login and registration is made of only firebase calls, and starting activity testing would require mocking tests.

LeaderBoardFrontEnd is responsible for reading and writing to firebase,

LeaderBoardCustomListAdpater takes in a list and and uses row.xml to generate the view that user see's of a scrollable list of highscores.

swipeTest and swipeViewAdapter and DemoFragment create the tab view of highscores that user see when they press Leaderboard on the page navigation.

- What are the most important classes in your program?

The most important classes in our program are leaderboardfrontend, leaderboard controller, GameActivityViews. BoardManagers for each game as they control the logic behind the game.

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

For the leaderboard we reduced the number of classes, and the navigation for the user. We did this by using a tabview which allowed the user to swipe through the various highscores. Additionally, we implemented a factory design pattern initially for the various leaderboards, but realized that it could be simplified to one class which is responsible for reading and writing to our database. And one class for sorting and adding new highscores to a local list.

For sequencer, the observer pattern was used extensively. The problem that it solved was that when the Movement Controller processed a movement, it changed the position within the sequence, or it registered a wrong move which means that the game must be terminated, but there was no clear way on how to connect these changes to the game activity. This was solved using the observer pattern so it cold communicate with the activity and call and update() method which took care of checking for various things such as if the game is over, updating the scores and the positio.

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

In each game activity we create an instance of a controller, in the controller we create an instance of LeaderFrontEnd, this class has a method called saveScoreToLeaderBoard. This method is called whenever data is changed in the firebase database using an event listener. Once called it saves the firebase data to leaderBoardController which adds the new high score using certain game logic conditions. This process is used for all games. The highscores are stored in firebase in a branch called Games. We display the data using a tab view. This works by first extracting the highscores of each from firebase, and storing them in leaderBoardReader, from there using fragments we display certain scores on the corresponding page.

The scores are displayed from lowest to highest to signify rank, and since rank is always lowest to highest we followed this structure for all our games.