# Script ( feel free to add and edit where you feel like ):

Start :

Setup :

Firebase open on one laptop near them - we present on the projector with Abdullah's phone connected

Marco: intro - Hi all …. (names + project ) pass it on to Jeff

Actually Jeff do both: Demo (Jeff "narrating" the tour):

Let's take a walkthrough of the app. After I finish downloading it and launch the app, I come to this screen. If my name is Jack, I would login or register as needed, using jack@gmail.com and the password 123456. Now that I'm in, I can pick three games to play.

I want to play sequencer, so I tap Sequencer and it brings me to another screen. I click new game and the goal of the game is to memorize a sequence of patterns. The score is decremented after every round u pass. After a while, if I decide to leave the game, and if I want to save it to play later, I can save the game, and the load saved game later. Now, I want to play another game, so I pick tictactoe, where it brings me to a screen where i can pick a difficulty, I pick player vs random, and play a game vs the random AI. I want to check the scores of how I did in the past games I played, so I now go to any starting activity and hit the leaderboard, where I can scroll through the leaderboards of each game. (go back to starting activity of sliding tiles)

Another new feature we added for sliding tiles was that it is now always solvable and cool small feature we added is that the background changes on every tap of the background.

TICTACTOE:

So for tic tac toe, several of the methods and classes were the same or very similar to the sliding tiles and sequencer games. However, since Tic tac toe was a game that was made for two players to play, there was also a few noticeable differences from sliding tiles and sequencer too. Now, tictac boardmanager had to have methods to check for different players turns. Another feature that was different in the game activity and touchmovement controller in tictactoe was that there was a countdown timer controlled by the two classes, which had the score based on the timer. Strategy was something that was completely different from the other two games, where depending on which game was played, a strategy could be called for the second player to use.

The refactoring component for tic tac toe was much more difficult then at first glance, sliding tiles was designed to work with only one player, and our tic tac toe incorporates not only player

versus player but player versus 2 types of AI, easy and hard. This made many of the pre existing classes from sliding tiles not applicable in this situation; For example:

- Different winners (and how to deal with that on highscores)
- Different moves, ie player 1/player 2 colours
- Different AIs (Sliding tiles didn't even have one)
- Different end results, ie player 1, player 2, tie, time, etc.

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.

Design

- The Game activity will take care of most of the visual stuff for this game. It calls the Speak function which will light up the tiles in the correct order depending on the sequence and the current round.
- The manager takes care of the background game logic, such as knowing if the game is over, checking whether a tap is valid, keeping track of the round.
- The movement controller process taps. It uses information from the manager to know how to handle each tap and in turn changes the information within the manager. If the board is “speaking” (showing the sequence) don’t do anything. If the tap is invalid, set game over. If the tap is valid, carry on.
- This all communicate back to the game activity using the observer design pattern. Whenever the manager gets changed, the game activity performs a few checks: If the game is set as over, save the score and exit the activity. If the round is over, speak.

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.

Tic Tac Toe Design:

- There is a strategy abstract class which allows for the player vs player or player vs AI

- Depending on which game mode is dependant on which strategy class extends to the abstract strategy class. Changing turns is important for tic tac toe, where the moves and checking who wins all depends on who's turn it is; classes used for strategy are :TicTacStrategy (the abstract class), TicTacEmptyStrategy, TicTacRandomStrategy and TicTacMinimaxStrategy
- Similar to the sliding tiles game, the tic tac toe game was designed with the same type of gridview
- The TicTacBoard keeps track of the variables for keeping track of the game_state (whether it is over or not{True, for over, false for not over}), the current players, and other things like that. TicTacBoard builds the board for the game, as well as the board iterator
- TicTacBoardManager class has all the methods that are called on the board and manages the board. Methods in this include touchmove, which processes the touch at certain positions, change turns, for changing the turns between players
- Timer class used to calculate score

LeaderBoard:

Structure:

- When a user presses LeaderBoard on the navigation page they are directed to a tab view, where they can swipe through multiple highscores for each game
- 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.
- The tab view is composed of three classes swipteTest, DemoFragment and swipeViewAdapter which allow the user to navigate through the tabviews;

  - Maybe mention something about how we didnt rename the classes, should be called swipeView and swipeFragment.

  - swipeView is the new intent whenever we move over to the leaderBoard screen; We initialize a swipeViewAdapter and a basic pager in order to keep track of the different pages, which are accessed simply by swiping from side to side.

  - Each time we enter a new page, including the first, we create a new swipeFragment, containing the information to be displayed on that current screen; This includes the various sizes for sliding tiles per user and per global, as well as tic tac toe and sequencer.

  - This new fragment is created by passing in items as a bundle when we create the fragment, and this fragment changes the existing xml file according to what lists/highscores type to be displayed. This means we only need 1 xml file for our swipeView.

- Maybe mention something about the slight error in refreshing the fragments.

- Mention something about how the fragments directly next to the current one are loaded, and thats what causes the bug. And maybe how to go about fixing this bug, what ive tried to fix it as well.

- When a user completes a game with a highscore the LeaderBoardFrontEnd's saving method is called which saves data to an internal LeaderBoardController which has it own temporary storage of called UserScores which is composed of Score objects.

This implementation allows any game to be added, further we had designed the LeaderBoardController using a factory design pattern originally but decided this implementation was simpler since it could be extended to any game and the code in the factory design pattern was largely similar.

# Questions

- 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.

# Guidelines:

Hi everyone,

We're happy to announce a Phase 2 extension until Friday 30 November at noon, with the usual late policy. Good luck with the end of term madness.

Please do plan your time carefully. You need time on the last weekend to prepare for your presentation, so we will mark the version that is in your repositories at the Friday deadline. **Your team should plan to have a rough draft of your program ready at least the day before the Friday deadline.**

Please re-read the assignment handout.

We understand that some of you will be nervous about the presentation and that this in fact may be the first presentation of your undergrad degree. To try to relieve some pressure, we're happy to note that the average grade for Phase 2 over the last several terms has been in the high 70's. A year ago, 84% of the teams earned a 70 or higher. That year, there were no failures, and, out of about 100 teams, only one team earned a mark in the 50's.

Here is further information about the presentations and marking process.

Teamwork: TEAM.md

This, README.md, and any meeting notes are the only files that should be pushed after the Friday deadline.

The TAs will have access to detailed git log information, including who contributed what. **In TEAM.md, please identify the usernames that appear in the git log.** The TAs may ask for details about any unbalanced commits. **Please also explain what each team member contributed.**

Presentation

Each presentation slot is 30 minutes. We will have a projector in the room with HDMI and VGA cables. Please of course bring a laptop to present from.

Here is an overview of the 30 minutes:

- 3 minutes for setup and introductions (please have your emulator already running when you enter the room)
- < 5 minutes to demonstrate your game centre
- < 10 minutes for a code walkthrough
- 5 minutes for TA questions
- 2 minutes to pack up
- 5 minutes for private TA discussion

Taking more than 15 minutes would mean that later teams would all have to wait, and an incomplete presentation will earn a lower mark. It's impossible to recover the schedule if we fall behind!

# Introductions

Each team member will briefly introduce themselves. Please include your name and say a sentence or two about which parts of the game centre code you worked on. This should match the contents of TEAM.md. Try to say it confidently — look at the TAs, and perhaps smile if you feel like it. It makes a good impression. One of you should also say what the group number is.

Game Centre Demo

Spend no more than 4 or 5 minutes demonstrating your game centre. You should have completed enough games to have some information in the scoreboard — or you might create it programmatically as the game centre launches, or perhaps with a button press.

We would like to briefly see your two new games, so please have them ready to go. We don't need to see sliding tiles. For the two games, please have saved games ready that are nearly complete so we can see the interaction with the high scores.

Unit Tests

Spend one minute running your unit tests to show us the coverage.

Code Walkthrough

For the remaining 9 or 10 minutes, you will present your design and show us around your code. Please note that this is not a lot of time, and **we expect you all to have practiced your presentation**. The presentation quality factors into the mark, so please make an effort to be clear and concise.

- You are welcome to prepare a UML diagram. If you do, spend time organizing it so that related classes are together. It can be a nice way to show the structure of your program.
- In your Java code in Android Studio:
    - Explain your scoreboard implementation, including how you store and manage the user scores.
    - Explain your design pattern code.
    - Briefly show us your best unit test class.
    - If you have other code that you're proud of, show us.

- While you present the code, we'll take notes about the packages you created, code formatting, Javadoc, naming, whether you used static inappropriately, any convoluted code, and any obvious code smells.
- After your presentation, the TAs will spend 5 or more minutes asking you about your code.

Prologue

After you leave the room, the TAs will spend the remaining time taking more notes and agreeing on a grade.

# Rough marking scheme

- 2 of the 17 marks will be for code coverage.
- 3 of the 17 marks are for presentation quality.
- 12 of the 17 marks are for design and code quality.

Code coverage

- 0/2 if the coverage is less than 50%
- 1/2 if the coverage is 51–74%
- 2/2 if the coverage is higher than 75%

Presentation

- 3/3 clearly and concisely explained the required information within the time limits
- 2/3 explained the required information within the time limits, but the presentation had some noticeable rough spots
- 1/3 explained most of the required information within the time limits, but parts of the presentation was hard to follow
- 0/3 did not explain substantial amounts of the required information

Design and Code Quality

This will be given a letter grade based on the UofT grading scheme. All the points above weigh in the code walkthrough will be taken into account.

An A+ means that we — David, Lindsey, Paul, and the TAs — would all be happy to use your code as a model solution for the rest of the class.

We hope this helps!

The CSC207 Staff