Outline

-- begin presentation --

Introduction

- Group number 0648
- Names
- Contributions

Demonstration

- Game Centre
    - Show: Registration, Login
- Hangman
    - Show: New Game, Loading, Complete game, Scoreboard,
- Sudoku
    - Show: New Game, Loading, Complete game, Scoreboard

Unit Testing

- Run Unit Tests, Sort by Line percentage
- Show tests for a class with 100% coverage
- Discussion of class with 0% test coverage.(Ex: UserManager)
    - Deals exclusively with file reading/writing.
    - Requires mocking filesystem of an Android device. This is outside the scope of the course(as announced in an email).
    - Also: part of the model in our MVC structure. Thus, doesn't need to be tested

Activities - need not be tested; only control view code

Model - need not/cannot be tested; https://piazza.com/class/jky6hcko7pi5v6?cid=770

Scoreboard Implementation

- Storage of high scores
    - Games create instances of Score when they are completed.
    - They pass in userName and Value for the name of the user and the number of points scored by the user.
    - They call GameScoreboard's method addScore with the Score and the directory corresponding to the game. (Ex: Hangman.txt for the Hangman game)
    - addScore then deserializes the ArrayList of scores inside of that game's associated text file, adds the new score to that ArrayList, sorts the ArrayList, and reserializes it for later use.
- Display of high scores
    - Each game has an associated HS(High Score) activity connected to an XML file which is shared by every game's HS activity.
    - The scores are displayed using TextViews which are set to display all users' scores for that game when the HS activity is first launched. The

Textviews switch between the current user's scores and all users' scores when the button in the top right is pressed.

Most important classes

- **Classes not particular to any game are the most important**(Ex: LoginActivity, User)
    - Responsible for core functionality.
    - App is reliant upon them to work.
    - Cannot be easily removed(i.e. requires significant refactoring to remove) without breaking a large portion of the code.
- **Classes for specific games are less important**(Ex: HangmanGameActivity)
    - Can be removed with little to no refactoring.
    - The rest of the app's functionality (e.g: the other games) is unaffected by removal of these classes as these are localised to each specific game.
- **Testing classes vary in importance**. (Ex: HangmanGameTest)
    - Can remove testing classes and still have the code run
    - Unimportant in a final product where the code previously passed tests.
    - Important in cases where we know future development would take place. (this is not one of those cases, for now)
        - Tells us that our changes didn't break any code which previously passed tests and that it functions in all cases.

Some Important Classes (non exhaustive)

- Board
- BoardManager
- Login
- User
- UserManager
- RegisterActivity
- Score
- GameCentre
- GameScoreboard
- HighScoresActivity
- Move
- Movement and Menu Controllers
- UndoMoveList

Design Patterns

Model-View-Controller Pattern

- Specify responsibilities of each class
- Make code more testable and understandable

Comparator Pattern:

- For some games, the goal is to get the highest score but for others, the goal is the lowest score. Ex: Number of moves in Sliding Tiles or Sudoku.
- Comparator allows for each game to specify which scores are best,i.e top scores, using a game-specific comparator.

Iterator Pattern:

- A clean way of checking the correctness of the board in Sliding Tiles is to iterate over the board and check that each tile(besides the first tile) has a value higher than its previous tile.
- To do this implement this, we needed to make the Sliding Tiles board iterable this was done following the iterator design pattern.
- An Iterator was also used in the Sudoku Board Generator Class to iterate cleanly through a 2-D Array generated game board for the purposes of assigning ID values and create a Board Object.

Observer Pattern:

- Used to synchronize the view seen by the user and the model for the game.
- Game Activities observe the game Game classes (Ex: HangmanGame) such that when changes are made to the Game classes, the Game activities, i.e. the user's view, is updated.

-- end presentation --

-- Q&A Time --