Mobile Application Programming: Android

CS4530 Fall 2016
Project 3 - MVC Battleship
Due: 11:59PM Monday, Oct 24th

Abstract

Build a Model-View-Controller implementation of the game Battleship on Android. The application will host a list of games that are in progress or have finished, as well as a user interface to play a game.

The game itself involves two grids positioned over locations in the ocean. One grid belongs to the player while the other belongs to his / her enemy. Each grid contains 5 ships that can be positioned in a row or column of the grid and have lengths of 2, 3, 3, 4, and 5 units. Players take turns launching missiles into individual grid locations in their enemy's grid with the goal of sinking the opponent's ships. When a missile is launched, the player is told wether the missile "hit" or "missed". Some variations of the game also say on a "hit" if the ship was "sunk", meaning that all of the locations the ship occupies have been hit, but our variation will not give that distinction. While both grids will contain hits and misses, each player may only see ships that are in their own grid. The game is won when all locations that the enemy's ships cover have been "hit". See http://en.wikipedia.org/wiki/Battleship_(game)#Description for more information.

The game will function in a hot-seat style of play such that when a player has taken their turn a screen covering the grids will instruct them to hand the device to their opponent to take the next turn. This game ideally would be constructed to support multiple devices, allowing each player to use their own device and eliminate the need for this covering screen. This configuration will not be required in this project, but will be required in a future project.

Components

- **Data Model** object inheriting from java.lang.Object containing a list of game objects
  - Offers an interface allowing the user to work with game objects to **create**, **read** (review), **update** (play), and **delete** them. That is, the current state of a game as well as concluded games should be maintained in a data store for review and statistics purposes.
  - Persistence of the game state list is required. E.g. the list of games should be saved at appropriate times while playing and should be reloaded on app start. This includes in-progress games and games that have been finished.
  - Game objects implementing the logic of the game should make up the bulk of the model code:
    - Contains the state information for the current player's turn, position of ships in the two grids, and locations that missiles have been launched to for each player.
    - Offers a method to launch a missile from the current player to a location on the grid.
    - Based on the game state information, the current game phase can be obtained (e.g. starting, in-progress, player 1 won, player 2 won).
    - The classic game allows the user to place their own ships on the grid before the game begins. To simplify the UI requirements of the assignment, instead write code that creates random, but valid, ship configurations for each player when the game begins. E.g. ships have the possibility of being placed in both columns and rows, and should not overlap or fall off the edges of the grid.

- **Views** that offer access into the model. These should be well implemented and function perfectly, but do not need to be very visually appealing. Boxes with colored fill or circles to represent open water, hits, misses, and ships are adequate. The screens involved are:
  - *Game List Screen*: A screen containing a list view that allows listing of games in-progress or ended that opens the game when an item is tapped. The item should note if the game is in progress or if it has ended, who's turn it is in that game (or has ended), and how many missiles have been launched by each player. Games can be started from this screen by pressing a "new game" button or other appropriate control.
  - *Game Screen*: A screen showing a grid that contains the locations of the player's ships and the locations their opponent has launched missiles against them. The screen also needs to show a grid that lets them launch missiles against their enemy's ships by tapping a grid cell, and shows where they have launched missiles previously, including "hit" or "missed" information. Colored boxes or circles are enough to communicate this information. If your interface is this simple, prefer blue to indicate open water, grey to indicate a ship, white to indicate a miss, and red to indicate a hit. The screen should not show where the opponent's ships are. If the game has already ended, no further missiles may be launched, and the screen should somehow indicate the winner.
  - When played on a tablet, both of these screens should be visible at the same time, with the game list on the left and the game screen on the right. The item in the list representing the currently visible game should be highlighted in some way, and should update its statistics information as the game is played. Using MVC and listener objects properly should make this rather simple to implement.
- **Fragment** objects should organize the views, acting as controller objects:
  - When views require information to draw the UI, the Fragment should query the model for that information when requested to do so. When the user taps a grid cell, the view should indicate this to the Fragment, which in turn should ask the model to perform the "launch missile" action. Use direct method calls to communicate "down" the class hierarchy and listeners to communicate "up" the class hierarchy where appropriate. Following the MVC pattern, the model and view should be oblivious of one another, never interacting directly.
  - When validating if the launch missile action is allowed, the model should respond to the controller (Fragment) either by returning information from a "launch missile" method call, or by a listener call saying "missile launched at location X/Y and was a hit/miss". It should also indicate "game won" information in a similar way.
- **Extra Credit**
  - 10%: Implement the UI allowing users to *place their own ships* before the game begins
  - 20%: Create an *AI player* that plays against the user instead of the "hot seat" configuration. The AI must be smarter than just choosing random locations! Consider creating a grid of numbers representing the "goodness" of a particular location based on information already known about those around it. Then have the AI choose the location with the highest "goodness". Though not required to get the extra credit, the best AI's will be able to traverse a tree of possible game states, searching for the most favorable move by taking into account several possibilities, looking turns into the future.

Handin

You should hand in a zip file containing your project. To do this, zip the folder and deliver it using web handin or the handin command line tool in the CADE lab. Hand this zip into:

handin cs4530 project3 your_zip_file.zip