# Mobile Application Programming: iOS

CS4962 Spring 2016
Project 3 - MVC Battleship
Due: 11:59PM Monday, March 21st

**Abstract**

Build a Model-View-Controller implementation of the game Battleship on iOS in Swift. 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. 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. Each player may only see the ships that are in their own grid. Players take turns launching missiles into individual grid locations with the goal of sinking the opponent's ships. When a missile is launched, the player is told wether the missile "hit" or "missed". The game also says on a "hit" if the ship was "sunk", meaning that all of the locations the ship occupies have been hit. 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 they will hand the device to their opponent to take the next turn. A view controller who's content says "Give the device to your enemy." should cover the screen when the user has taken their turn. This allows the user to give the device to the other player without showing their own ships.

**Components**

- **Data model object** containing a list of game objects
  - Offers an interface allowing **access**, **addition**, **removal**, and **updating** (playing) of games
  - Persistence of the game states is required. E.g. the list of games should be saved at appropriate times while playing and should be reloaded on app start. Save the file in the Documents directory in the sandbox area for your app (code below to get the path).
  - **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 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 end 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. The screens involved are:
  - **Game List Screen**: A screen containing a table view that lists games that are in-progress or ended, and that opens a game when its row is tapped. The row 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 ships remain un-sunk for each player. Games can be started from this screen by pressing a “new game” button.
    - **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 another grid representing the player’s opponent and that lets them launch missiles by tapping a grid cell. This grid should show where they have launched missiles previously, including “hit”, “missed”, and “sunk” information. The screen should not show where the opponent’s ships are, for reasons that are hopefully obvious.
- Views should be organized by being added to **view controller objects**:
    - When views require information to draw the UI, the view controller should query the model for that information when requested to do so. When the user taps a grid cell, the view controller should ask the model to perform the “launch missile” action (using a delegation pattern or target-action mechanism), rather than the view doing this. Following the MVC pattern, the model and view should be oblivious of one another.
    - When validating if the launch missile action is allowed, the model should respond to the controller either by returning information from a “launch missile” method call, or by a delegate call saying “missile launched at location X/Y and was a hit/miss”. It should 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
    - 10%: 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 those around it. Then have the AI choose the location with the highest “goodness”.
    - 25%: Anyone interested in Bluetooth communication may review the 2011 class lectures on bluetooth communication and implement a bluetooth pair play version of the game on 2 devices.

### Documents Directory

Use the following code to get the path to the documents directory, then the path to a file therein:

```
let documentsDirectory: String? = NSSearchPathForDirectoriesInDomains(
    .DocumentDirectory, .UserDomainMask, true)?[0] as String?
let filePath: String? = documentsDirectory?.stringByAppendingPathComponent("file.txt")
```

### Handin

You should hand in a zip file containing your project. To do this, zip the folder by right-clicking it and selecting “Compress”, then handing it in using web handin or the handin command line tool. Hand this zip into:

handin cs4962 project3 your_zip_file.zip