Mobile Application Programming: iOS

CS4962 Spring 2016
Project 4 - Networked Battleship
Due: 11:59PM Monday, Apr 4

Abstract

Extend your Model-View-Controller implementation of the game Battleship on iOS to use an Internet-connected server, allowing two players to play against one another on separate iOS devices. 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.

As before, the game itself involves two grids positioned over locations in the ocean. One grid belongs to the player while the other belongs to his 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 be modified to be multiplayer over the network, and communicate with a server using a RESTful API. All game information and logic will be hosted on the server. Remove the screen asking the player to pass the device to their opponent. Instead, present UI showing the user who's turn it is, polling the server every 500ms to update that state. Do not allow the user to launch missiles when it is not their turn. If you wish, you may allow the user to return to the game list screen, allowing them to start additional games while they wait. Multi-game functionality is not required, but is certainly encouraged.

Components

- **Networked Battleship** has all the gameplay requirements of project 3 not replaced by features below.
- **Data Model** should be modified to communicate with a server at http://battleship.pixio.com
  - API specification is below. Must be capable of handling JSON data.
  - Ship placement and most game logic will now be handled by the server.
  - Games will be saved on the server, made available in a lobby, which replaces the local game file used previously. Game and player IDs need to be saved, however, to access in-progress games.
- **Lobby List View** now contains all requested games found on the server with a specified status (WAITING, PLAYING, DONE), selectable using an appropriate control. The new game control should still be shown. Rows representing games should now show at least the game name and status. Include other statistics about games where available.
  - Tapping the new game control adds a game to the server by creating a new game, first collecting the game and player name with appropriate UI. Then the game UI is presented to play the game
  - Selecting a game in WAITING mode joins the game, first collecting the player name with appropriate UI. Then the game UI is presented to play the game.

    - Selecting a game in the PLAYING or DONE modes shows a simple game summary, unless the game was started previously on the device (it will have saved the game & player ID). In that case, open the game UI and continue the game.
- **Game View** should be the same as in project 3. If you built project 3 using MVC, very little should need to be changed in your game view.

Extra Credit

- **My Games** 5% - Extra credit for adding a 4th lobby option for all games played on the device. The server does not offer this search, so you will have to piece it together using server information and game / player IDs saved from games as they are played. This allows the user to see all games in all statuses that they are involved in.
- **Multi-Game Functionality** 5% - Allow the user to be in several active games at the same time. Be sure to poll the server every 500ms for each game to determine who's turn it is. Display this in the game list screen as well as in-game.
- Extra credit will also be given for **substantially improved UIs**.

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 project4 your_zip_file.zip

Battleship Server API

All request and responses are in JSON and follow JSON guidelines. "{ }" represent a dictionary of Key:Value pairs (can also be thought of as an object, where keys are class variables), and "[ ]" represent an array.

The board is a 10x10 grid with rows and columns ranging from [0-9].

Response will use HTTP status codes.

- `200` - Request handled successfully
- `4xx` - User request was not handled because the user request is malformed, missing parameters, or attempting to access a resource that is not allowed.
- `5xx` - Server error. Please tell me what you did to receive this error.

In the case of a `4xx` or `5xx` error, check the `message` property in the top level of the JSON response.

Games

Get game detail

Get details for the game with the provided id. Id is a GUID.

Request: GET /api/games/:id

Response:

```
{
    "id": GUID,
    "name": STRING,
    "player1": STRING,
    "player2": STRING,
    "winner": STRING,
    "missilesLaunched": INT
}
```

Get game list

Get all games currently on the server.

Request: GET /api/games

Response:

```
[
    {
        "id": GUID,
        "name": STRING
        "status" ENUM
    },
    ...
]
```

Status ENUM: DONE, WAITING, PLAYING

- DONE: Game has finished. No more actions are allowed
- WAITING: Game needs a second player. Allowed to join
- PLAYING: Game is currently in progress. Cannot join.

Join the game with the given id

Join the game with the provided id. Id is a GUID.

Request: POST /api/games/:id/join

```
{
    "playerName": STRING
}
```

Response:

```
{
    "playerId": GUID
}
```

Create a new Game

Create a new game with the provided name. Game will be in a `WAITING` status until another player joins the game. Once a player joins, the status will change to `PLAYING`. The player who created the game will become player 1 and will be provided a GUID to identify the player.

Request: POST /api/games

```
{
    "gameName": STRING,
    "playerName": STRING
}
```

Response:

```
{
    "playerId": GUID,
    "gameId": GUID
}
```

In Game

Making a guess

Make a guess to try and hit a ship. The integer for `shipSunk` will represent the size of the ship sunk or `0` if none were sunk.

Request: POST /api/games/:id/guess

```
{
    "playerId": GUID,
    "xPos": INT,
    "yPos": INT
}
```

Response:

```
{
    "hit": BOOLEAN
    "shipSunk": INT
}
```

Whose turn is it

Use to poll the server and determine the players turn. Can be used when waiting for an opponent player to join the game, i.e. will return false for player 1 on a new game until an opponent player has joined. If the game is in progress, `winner` will be `IN PROGRESS`. If the game is over, `winner` will contain the name of the player who won the game.

Request: POST /api/games/:id/status

```
{
    "playerId": GUID
}
```

Response:

```
{
    "isYourTurn": BOOLEAN,
    "winner" : STRING
}
```

Get Players Board

Will return the status of the player's entire board for the game with the given id. The board is represented as a list of cells. `playerBoard` represents the players own board (ships and all), while the `opponentBoard` shows the current players hits and misses against the opponent.

Status ENUM: HIT, MISS, NONE

- HIT: a player has hit this cell
- MISS: a player has missed this cell
- SHIP: this cell is part of a ship and has not been hit
- NONE: this cell has no activity

Request: POST /api/games/:id/board

```
{
    "playerId": GUID
}
```

Response:

```
{
    "playerBoard": [
        {
            "xPos": INT,
            "yPos": INT,
            "status": ENUM
        },
        ...
    ],
    "opponentBoard": [
        {
            "xPos": INT,
            "yPos": INT,
            "status": ENUM
        },
        ...
    ]
}
```