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 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 whether 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.

Components

• **MVC Battleship** has all the requirements of project 3.
• **Data Model** should be modified to communicate with a server at http://battleship.pixio.com
  • API specification is below.
  • Must be capable of handling standard JSON data. Recommend to use GSON library.
  • Ship placement will be handled by the server.
• **Views** Games and players will be named and the UI must be updated to display those names.
• **Extra Credit**
  • Extra credit will be given for substantially improved UI’s.

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 cs4962 project4 your_zip_file.zip`
Battleship Server API

A very simple REST api for the CS4962 Android Battleship MVC assignment. 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:**

```json
{
   "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 player's 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

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

**Response:**

```json
{
    "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 player's own board (ships and all), while the `opponentBoard` shows the current player's 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

```json
{
    "playerId": GUID
}
```
Response:

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