BattleShip Project: Detailed Level Design

Server Side

# Module I/O Router

- RegPlayers: {socketID: player, socketID: player}:

An object that refers a unique user’s socketID to a single player object.

A player object is unique to it’s Game.

* An Ajax call to ‘/api/users/ returns a users.getUserList() result: a JSON array of users objects.

- socket.on("connection") ---> establishConnection():

* gives the user a socketID
* If the user exists in Users module in registeredUsers object: set user.isOnline = true
* If the user doesn’t exist in Users module in registeredUsers object: use modules method addUser() to register him.
* socket.on(“disconnect”) ---> pauseGame() / +eliminateConnection()
* socket.on("guess") ---> player.resolveGuess(clickX, clickY)
* resolveGuess: in Game Manager module in class player.
* player: the player object according to socketID (as translated from the RegPlayers objects);
* socket.on("requestGame") ---> fwdRequest(user)
* emits "requestGame" to target user
* socket.on("acceptGame") --->

1. registerGame(socketID1, socketID2)

* Registers players in RegPlayers objects

1. game.initGame()

* initGame: in Game Manager module in class game.
* Invokes game.player.initBoard() for each player
* Returns en objects with both’s players’ boards

3. Emit players’ boards repectively to their corresponding players.

* socket.on("gameRefused") ---> fwdDeclinedRequest(user)
* emits "gameRequest" to original user
* AnnounceTurn(player): emits(“turn”) to player socketID

# Users Module

* Users: {nick: user obj, nick: user obj, nick: user obj}
* getUserList(registeredUsers)
* addUser(nick)
* setOnline(user)
* setOffline(user)
* **Class User** {

- socketId: null

+ Method: AcceptGame(user) ---> returns a boolean value.

}

# Game Manager module

* **Class Game** {
* gameID: number/ string
* player1: new Player(this)
* Player2: new Player(this)

+ Method: isGameOn() ---> returns a boolean value.

+ Method: initGame() ---> returns an object with 2 initialized player.array (of each player)

+ Method: setTurn(player) ---> calls I/O Router.announceTurn(player);

+ Method: isWon() ---> returns a boolean value.

}

* **Class Player** {
* game: the Game object that created the Player
* User: reference to the player’s user object
* opponent: null
* hitCount: number
* Board: []

+ Method: setOpponent(Player object)

+ Method: resolveGuess(opponent.board)

+ Method: initBoard(this.board)

}

Client Side

# Connection/Authorization Module

* displayLoginWindow() - on load
* validateNick(nick)
* Button “click” listener ---> server communicator module.connectUser

- respondToGameRequest(user) -> returns boolean

# User Input Module

* getCellClicked() ---> returns {x:x, y:x}
* moveShip(ship, x, y)

# Server Communicator

* on(“gameRequest”, user) ---> calls CA.respondToGameRequest(user)
* sendRequest(user) ---> emits (“gameRequest”, user)
* getOnlineUsers() - sends an Ajax request to server
* connectUser(nick) ---> socket.connect
* sendGuess(x,y) ---> emit(“guess”) {x,y}
* sendReponse(response): emit("acceptGame") or emit("gameRefused")

# Game Manager

* localBoard: new Board
* targetBoard: new Board
* Event listener: board. Click
* validateCell(x, y, board)

# Drawer

* **Class Board** {
* boardCells: [100 Cell objects]

+ Method: initTargetBoard(board array)

+ Method: initLocalBoard()

+ Method: DrawBattleShip(x, y, direction, length)

}

* **Class Cell** {
* status: string

- x: num (refers to 2d board array cell index)

- y: num (refers to 2d board array cell index)

+ Method: DrawEmpty()

+ Method: setHit()

+ Method: setEmptyGuess()

}

* messageBoard: object
* messageBoard.displayMessage(msg);

BattleShip Project: Development Phases

# Phase 1: Basic routing and user handling

* User login process on client-side
* Inserting a new user on server
* **Retrieve available user list by request**

Estimated time: 5 hours

# Phase 2: Game requests

* On user’s choice from a list, the server initializes a new game between both users
* **Both users receive a message “you are connected with {user}”**

Estimated time: 6 hours

# Phase 3: Loading game/display

* Server sends hardcoded it to them correspondingly
* **Both clients display boards correctly**

Estimated time: 8 hours

# Phase 4: Turns

* Server selects randomly the client to start the game
* **The selected client receives the turn and can start emitting guesses**
* **The other client cannot emit guesses when turn is not his**

Estimated time: 8 hours

# Phase 5: Game Logic

* Server checks for a winner in every round
* **Both client display cell status at the corresponding board**
* **The right client gets the “turn” token**

Estimated time: 8 hours

# Phase 6: Win/Lose

* **Both winning/losing clients receive a message**
* **Clients can then choose to play another game**

Estimated time: 16 hours

# Phase 7: Prettifiying

* **Making the whole thing look nice**

Estimated time: 14 hours

BattleShip Project: Flow Charts

# Implementation of the turn process:

1. Client emitting guesses
2. Server processes guess and emits the cell index and its new status
3. Both clients display the newly sent changes
4. The server emits “turn” to the right client, making only them able to emit a new “guess”