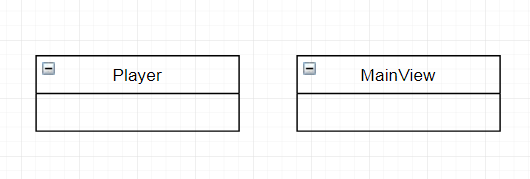
# DCU School of Computing  Assignment Submission

|  |  |
| --- | --- |
| **Student Name(s):** | **Aidan Fogarty, Eoghan Murphy, Michael Dowling, Oishin Smith, Daniel Perecz, Martin Zaloudek** |
| **Student Number(s):** | **16413106, 16368691, 16473694, 16401096, 16499124, 15551553** |
| **Programme:** | **BSc in Computer Applications** |
| **Project Title:** | **CA314 Assignment 1 - Analysis** |
| **Module code:** | **CA314** |
| **Lecturer:** | **Renaat Verbruggen** |
| **Project Due Date:** | **26/10/2018** |

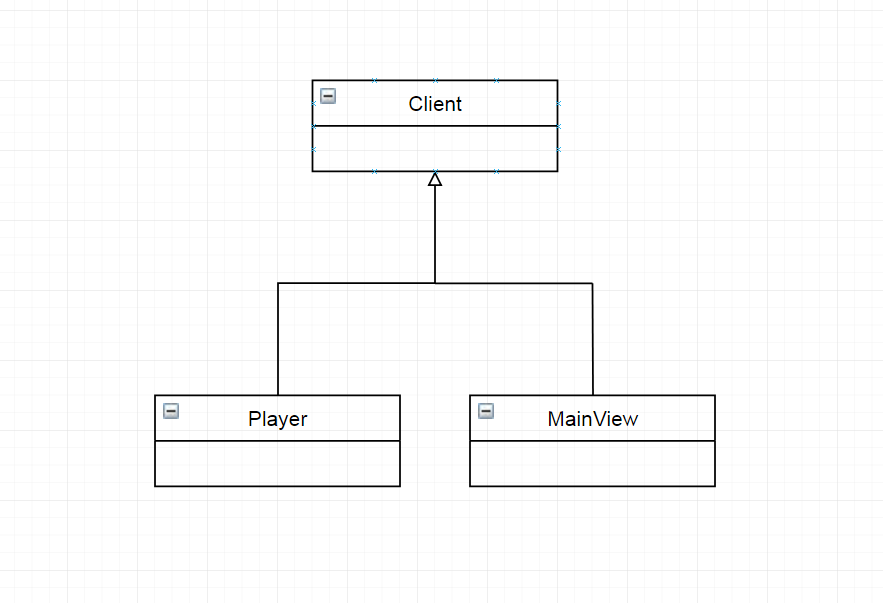
|  |
| --- |
| **Declaration**  **I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.**  **I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.**  **This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at**[**http://www.library.dcu.ie/citing&refguide08.pdf**](https://websvc.dcu.ie/%22http://www.library.dcu.ie/citing&refguide08.pdf/%22)**and/or recommended in the assignment guidelines.**  **I understand that I may be required to discuss with the module lecturer/s the contents of this submission.** |

**Refined class diagrams**

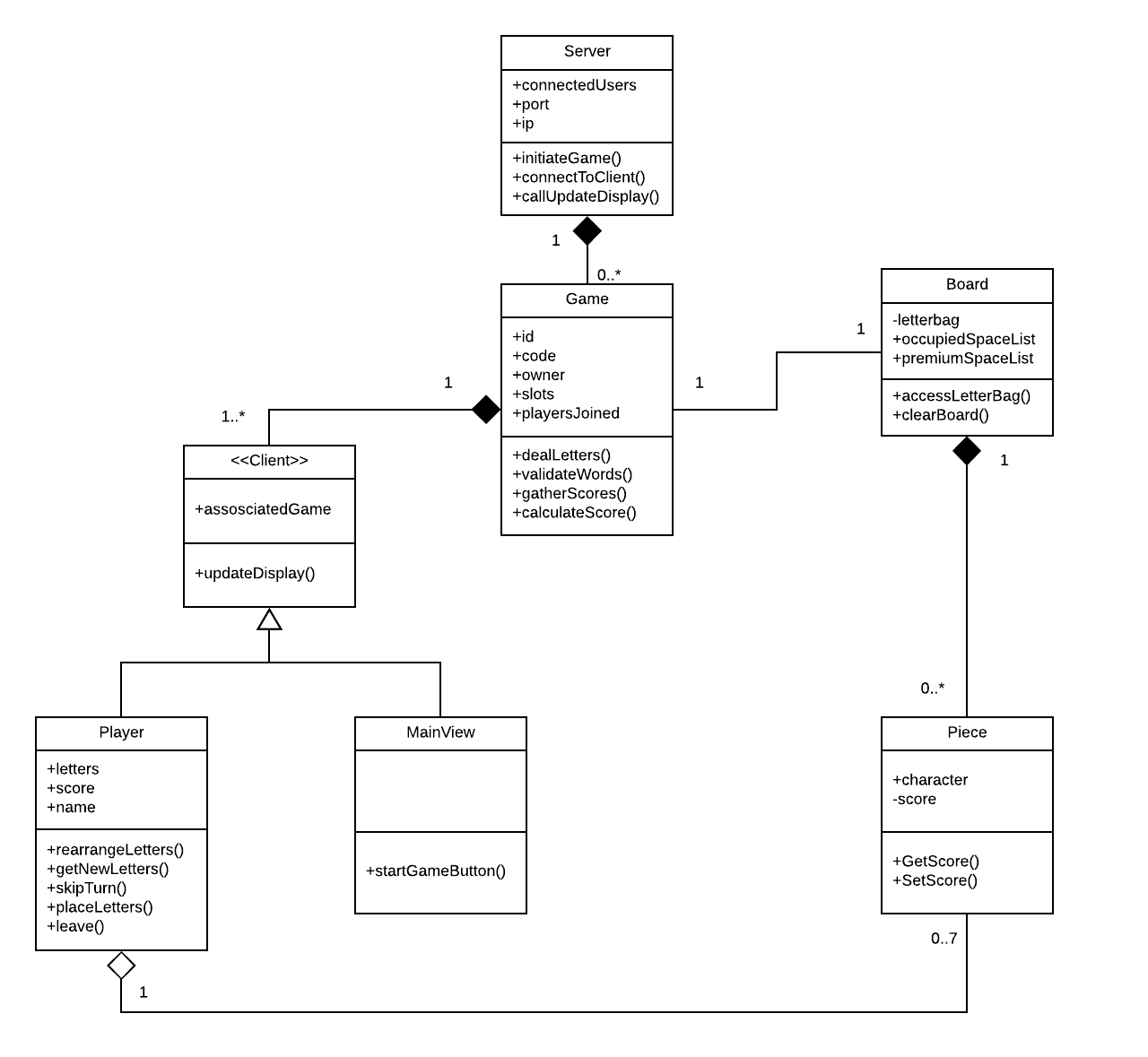
Our original system had “Player” and “MainView” as separate classes



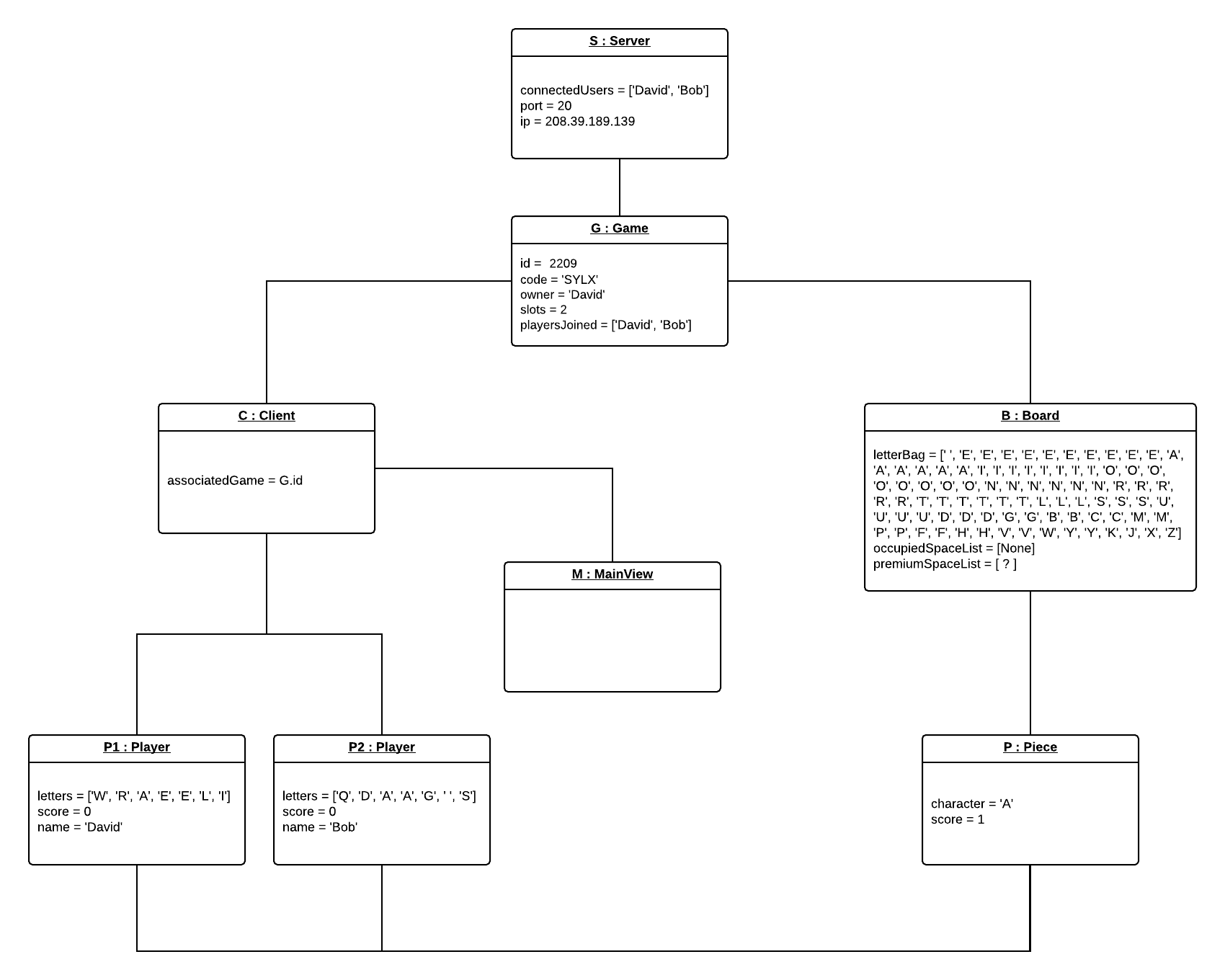
We changed this so they both inherited from a “Client” class to avoid repetition in our code.



This required changes in our class diagram and use cases but these were done before the last deliverable and as such are not documented here



**Object diagrams**



**User interface mock-ups**

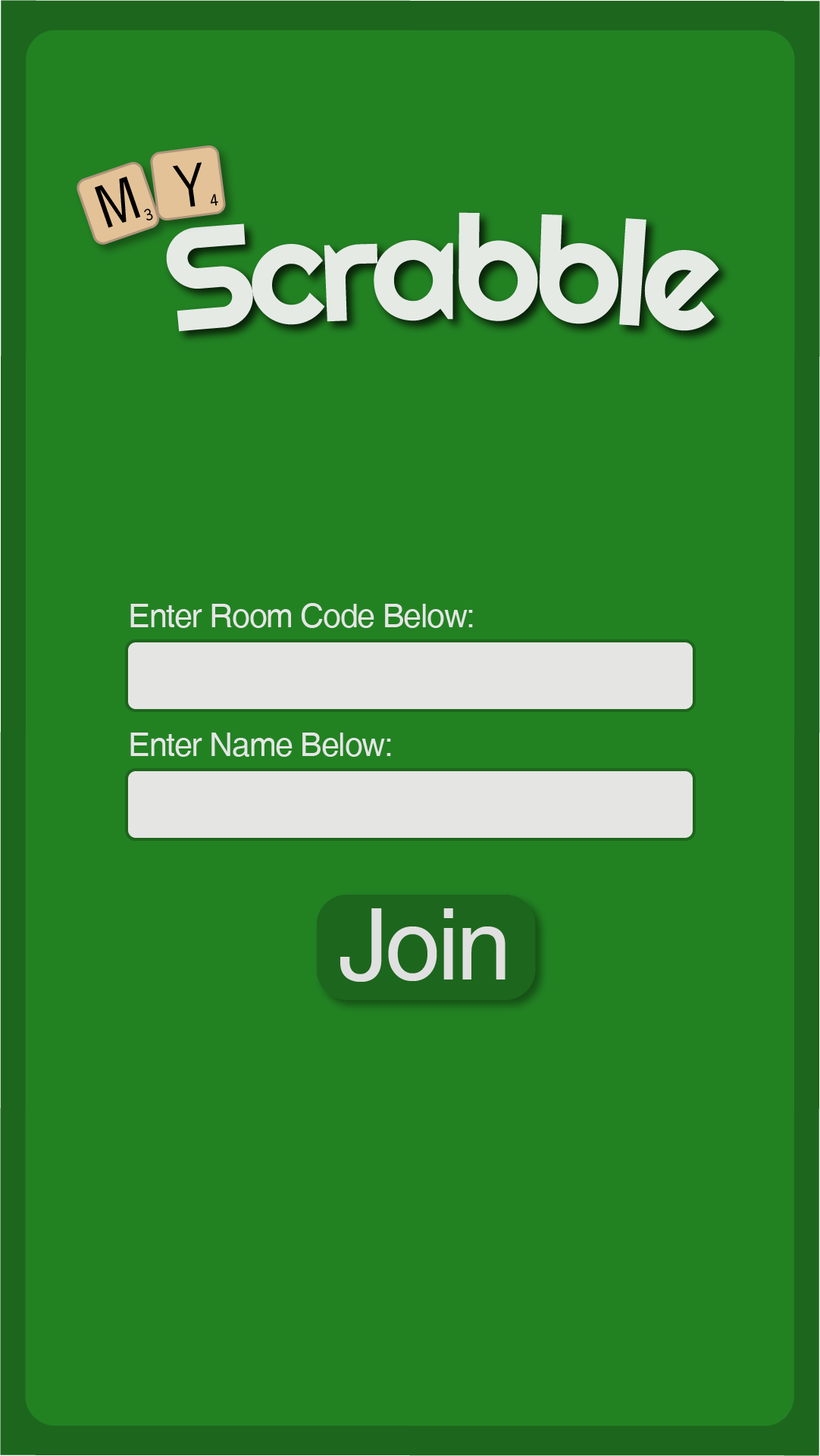
MyScrabble main menu:



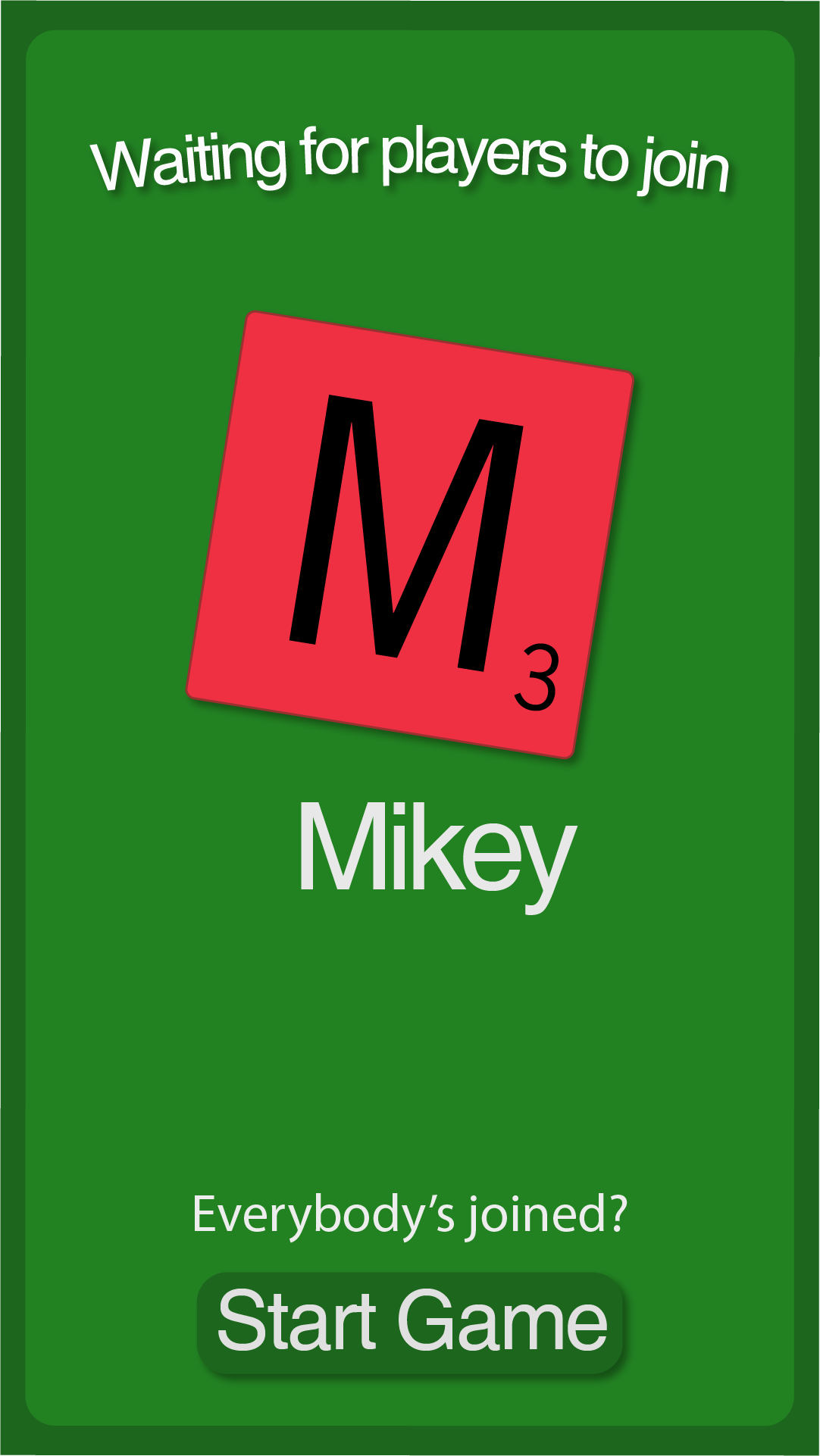
MyScrabble join screen (Main View):



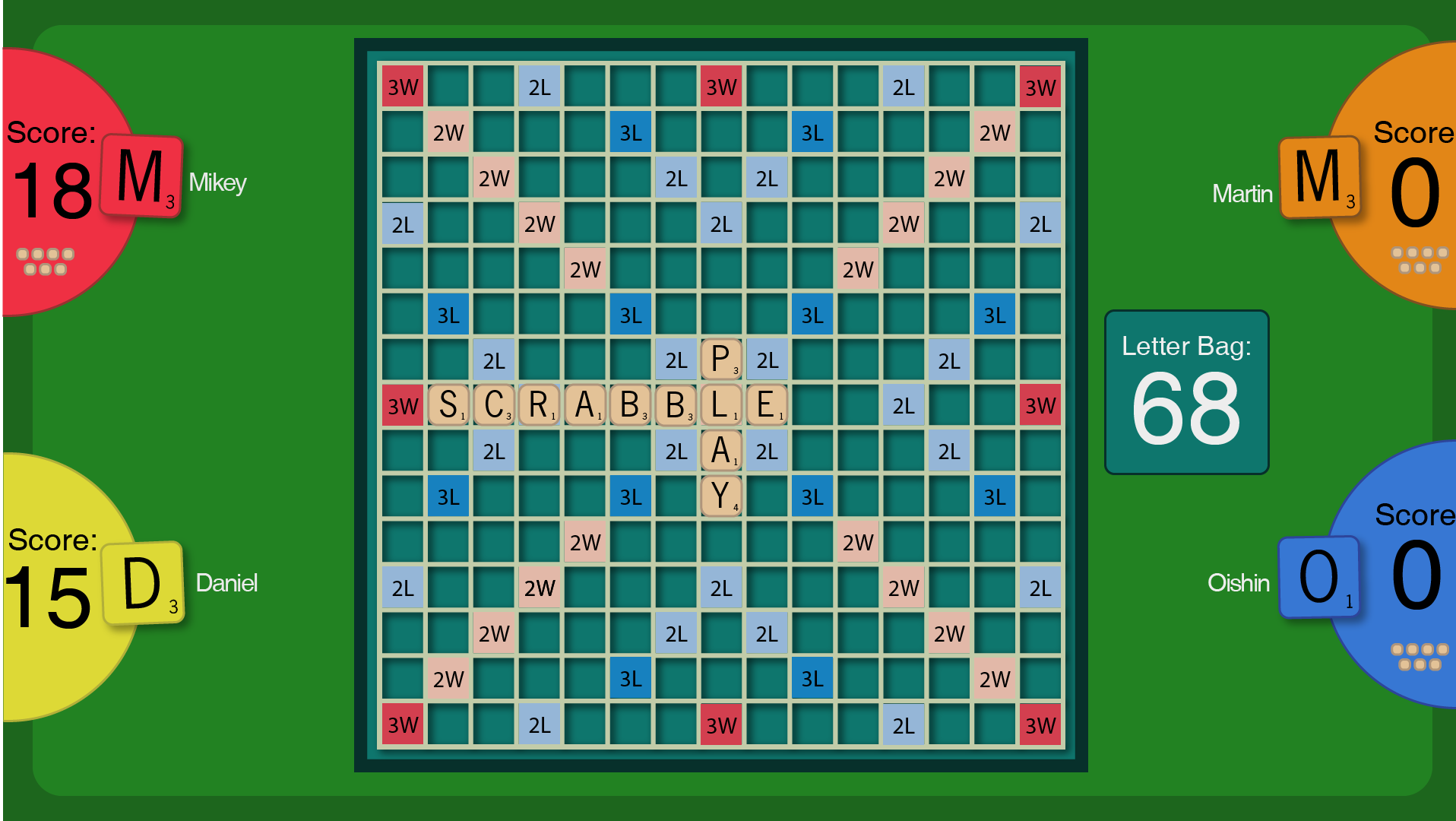
MyScrabble Join Screen (Player):



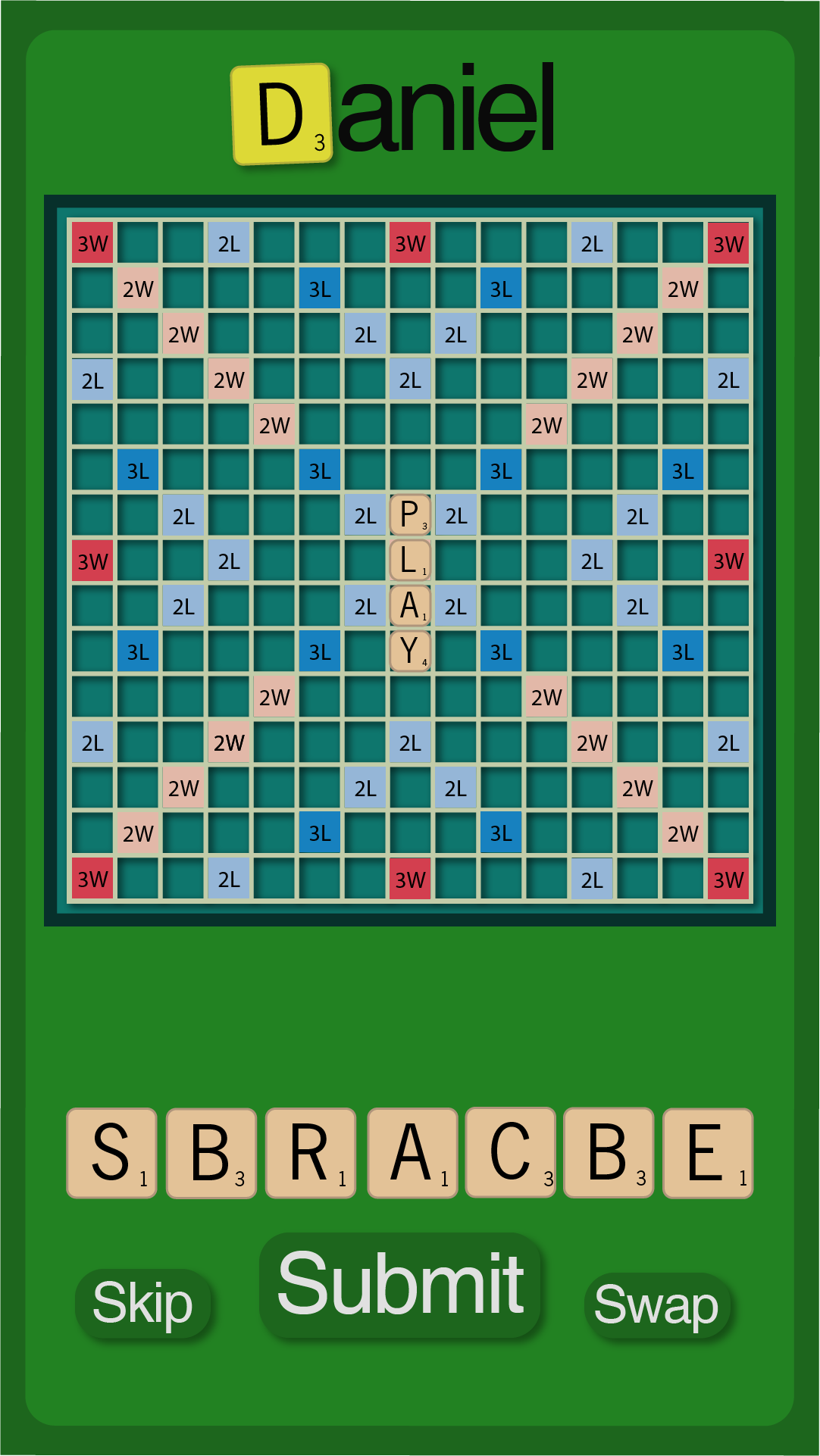
MyScrabble waiting screen (Player):



MyScrabble game screen (Main View):

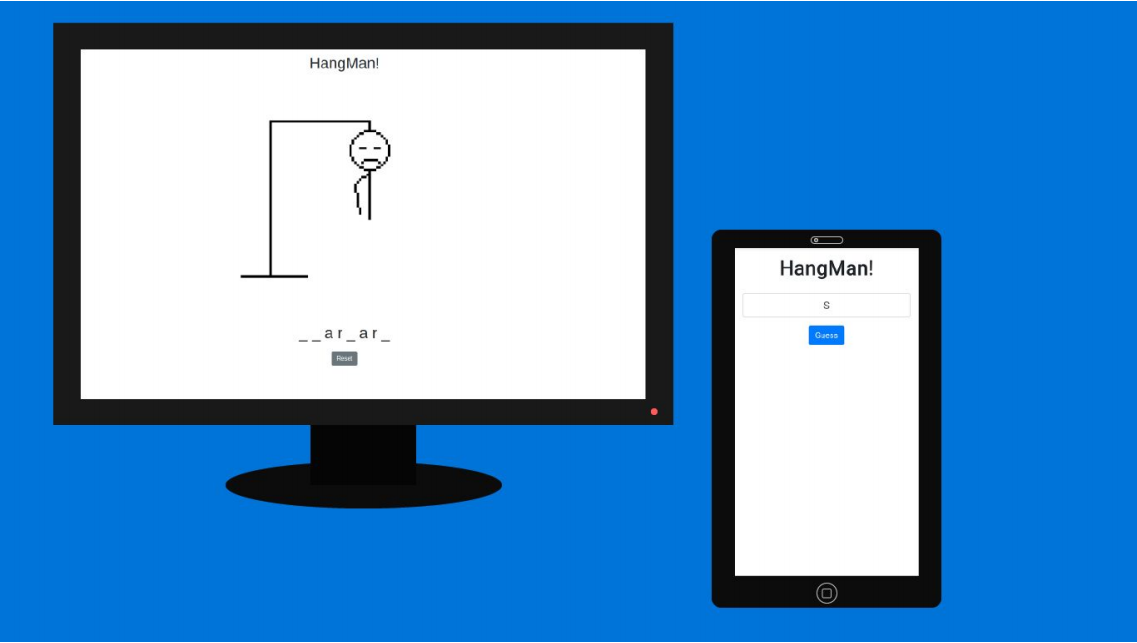


MyScrabble game screen (Player):



**Client-Server Experiment: HangMan**

We decided early on that we needed to build a prototype to test our novel idea. After a bit of searching, we decided on using Node.js with Socket.io to build it. We learned the basics with a few tutorials and chose to build a version of Hangman as the actual gameplay would be very easy to code (just check if a letter is in a word).



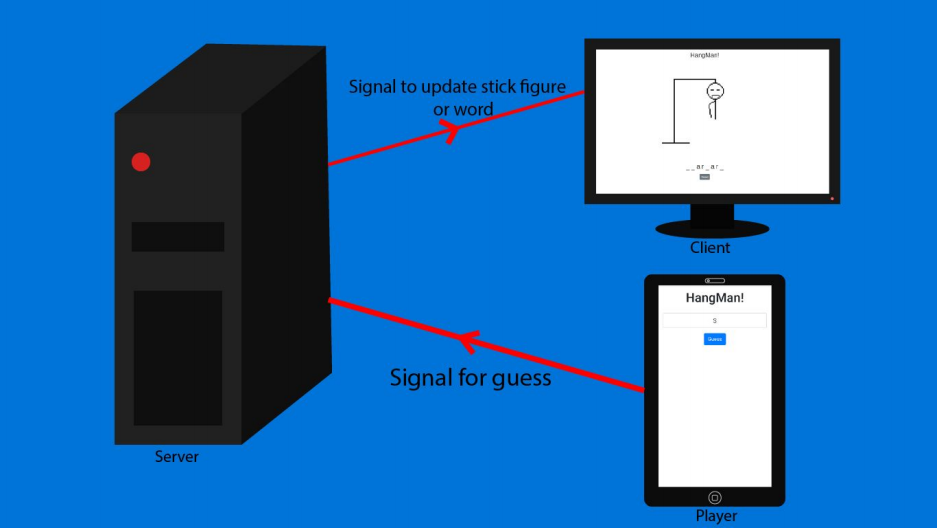
How the game works, more devices can be connected.

This version of Hangman used one device as a client and allowed many other devices to connect and guess letters. Much like a Jackbox Party Pack game, it allowed for any device with a web browser to be used to play. It was a basic idea and didn't really make sense as a multiplayer game but it was more a proof-of-concept and the game itself is instantly recognisable.

**Implementation:**

Thanks to Node.js the code itself turned out to be quite short. There were three main files, a HTML file for the client view, another for the player view and a JavaScript server file to serve both pages and handle most of the logic.

Using Socket.io and a bit of jQuery we could allow a player to guess a letter from the player view, this would, in turn, emit a signal to the server to process that guess and, depending on the result, a signal would be sent out to the client view. This signal would either cause the word displayed on-screen to be updated or would display more of the stick figure on the main screen to illustrate the player had lost a life.



A diagram explaining the relationship of the client, the player and the server.

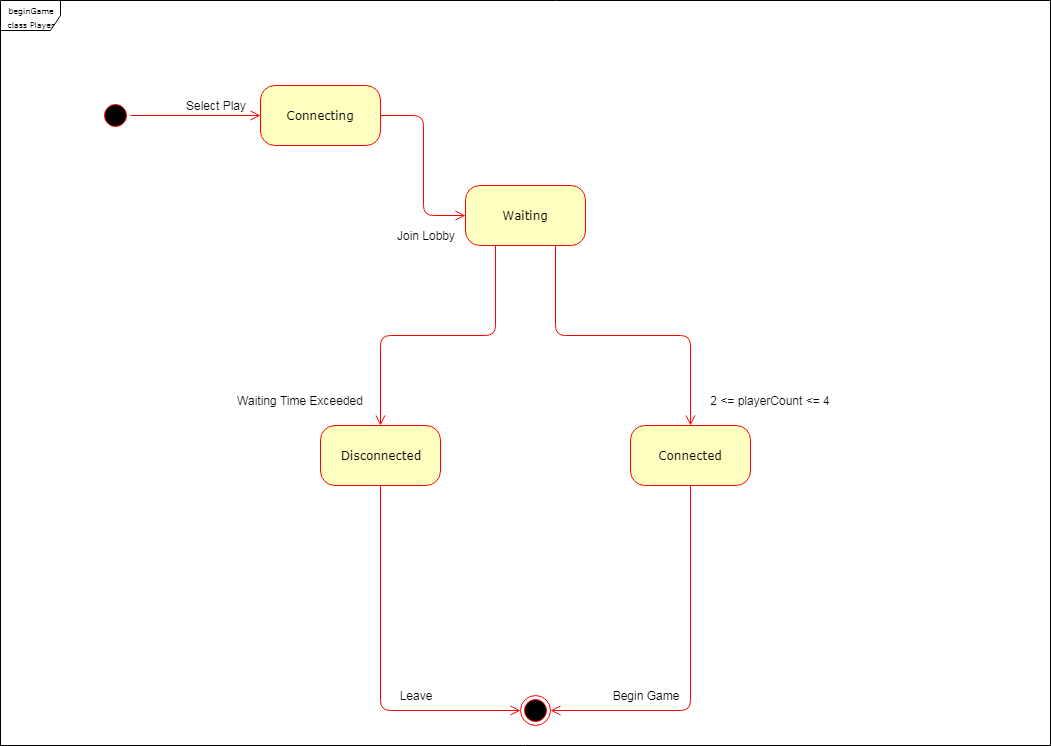
We could have added many extra features. Currently, the word you have to guess is hard-coded into the game and previous letters guessed don't appear anywhere on-screen.

All of these bells and whistles are outside the scope of a simple prototype. The purpose of the game is just to display how Node.js and Socket.io can be used to create a dynamic web version of a multiplayer game, which we think it does well. This will help us with our project.

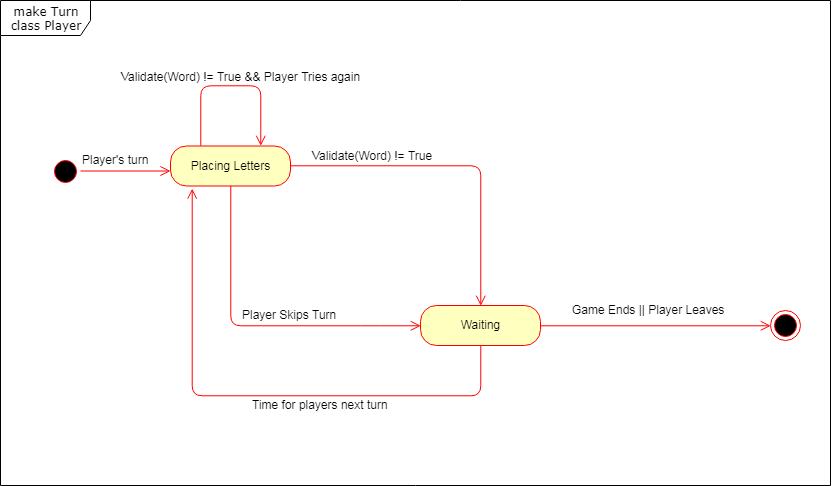
Link to the prototype’s git repository: https://github.com/Eoghan-Murphy/HangMan

**State Machines**

**Begin Game (From Player’s POV):**

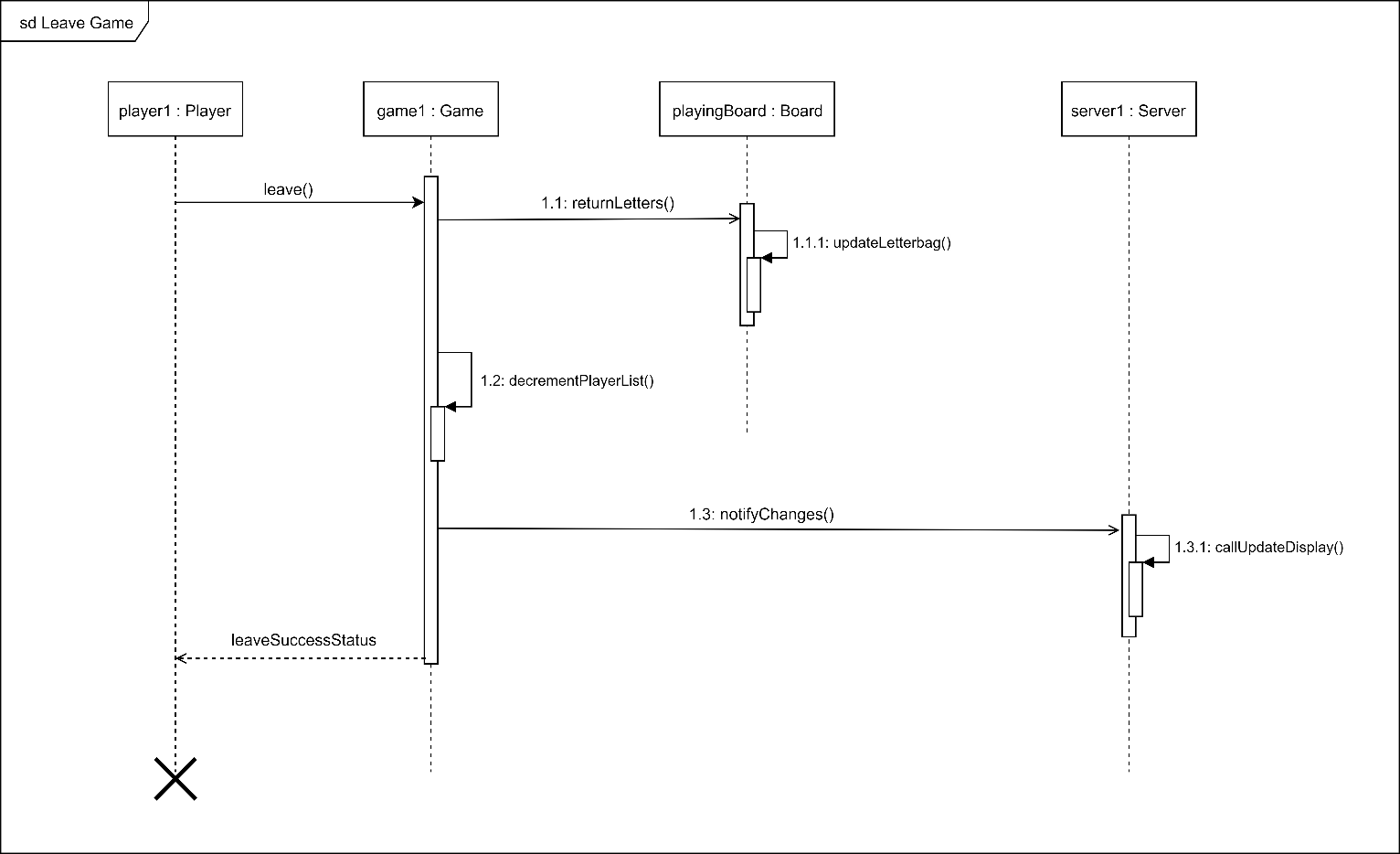


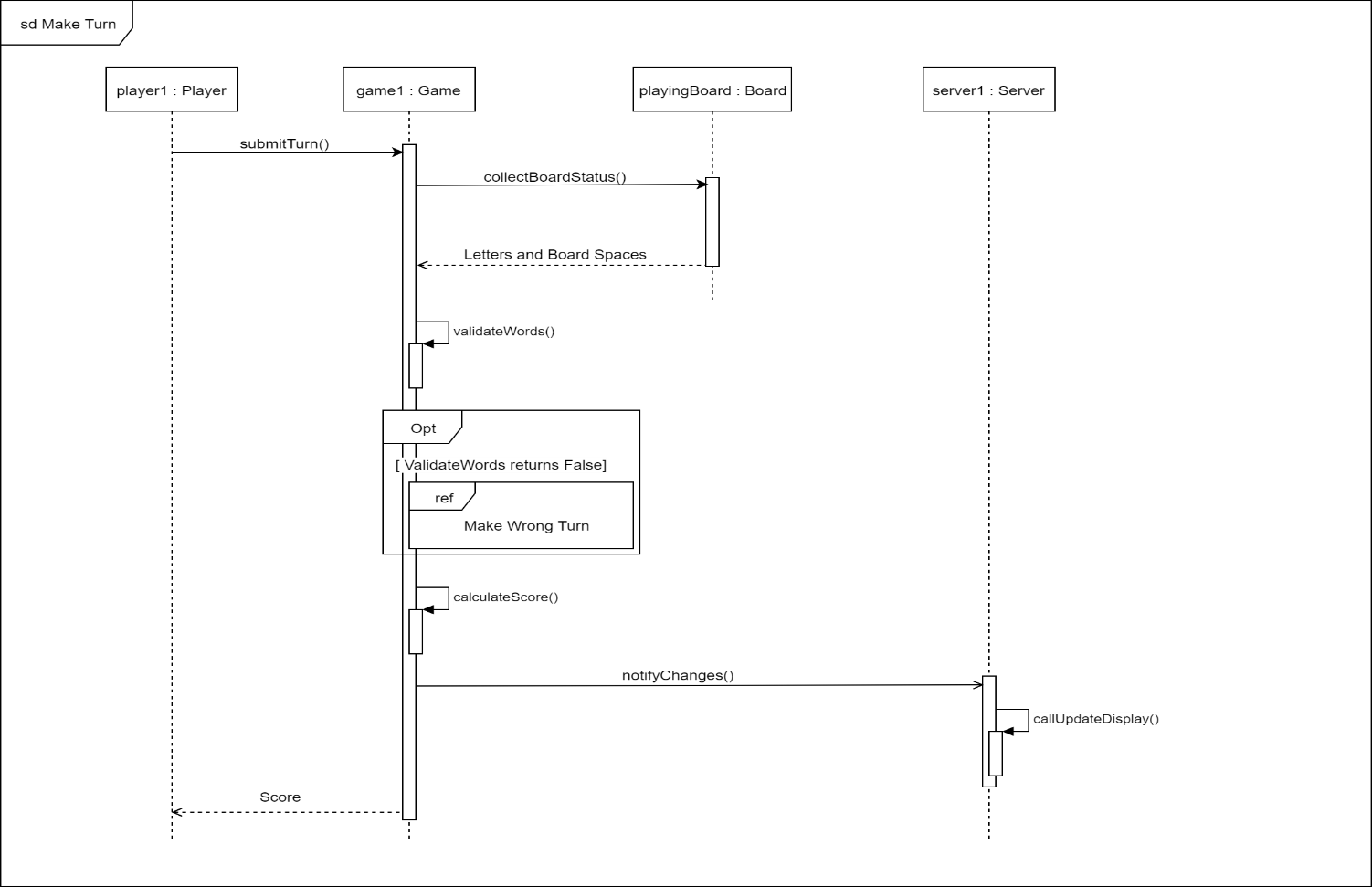
**Make Turn (From Player’s POV):**



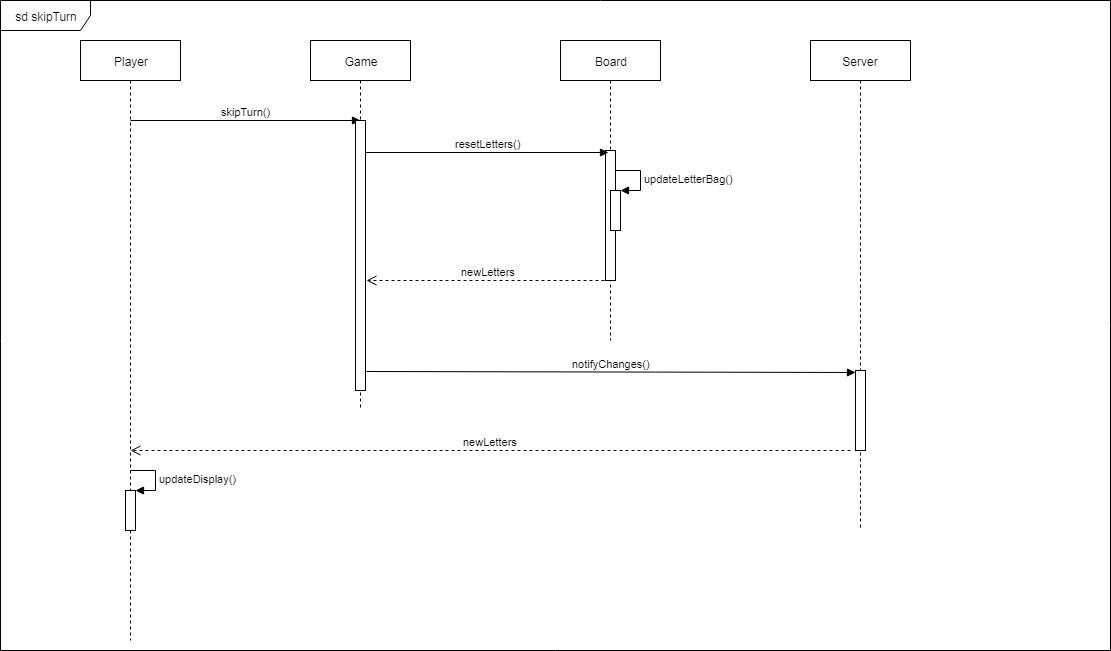
**Sequence diagrams**

**Leave Game:**

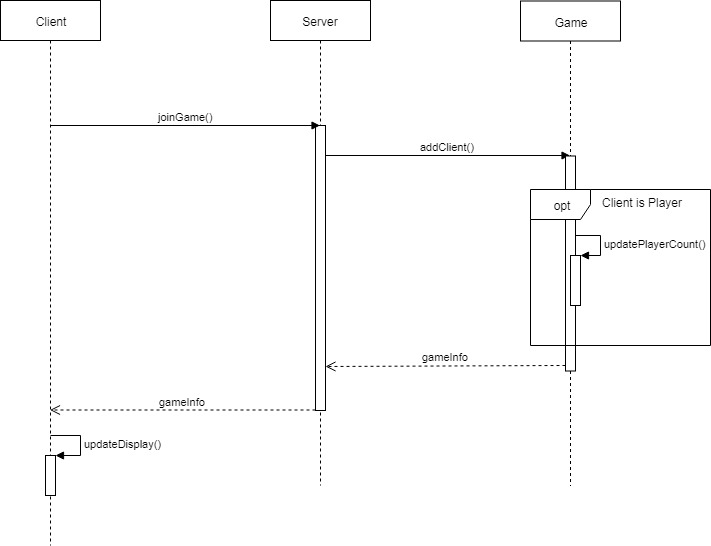
 **Make Turn:**



Skip Turn:

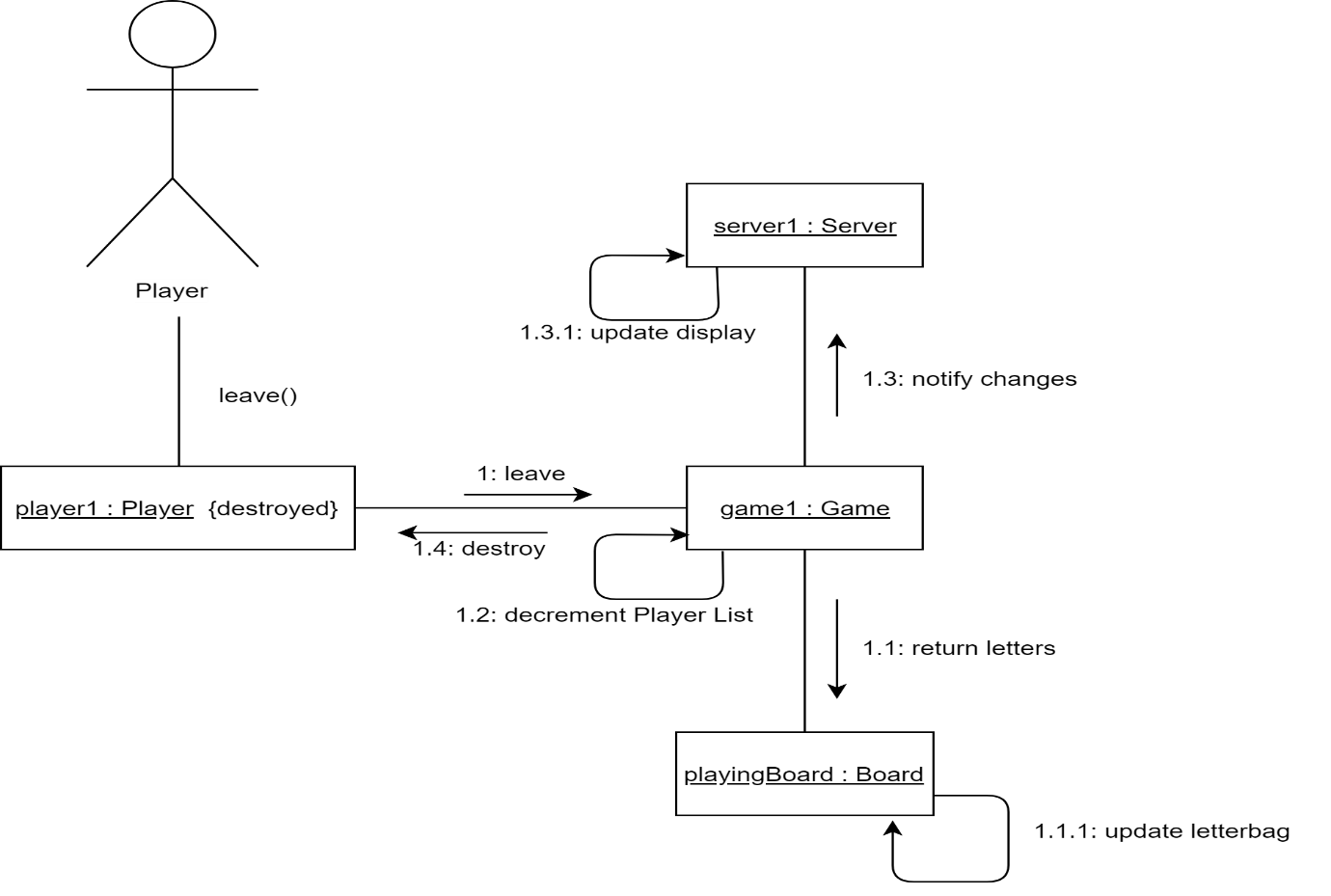


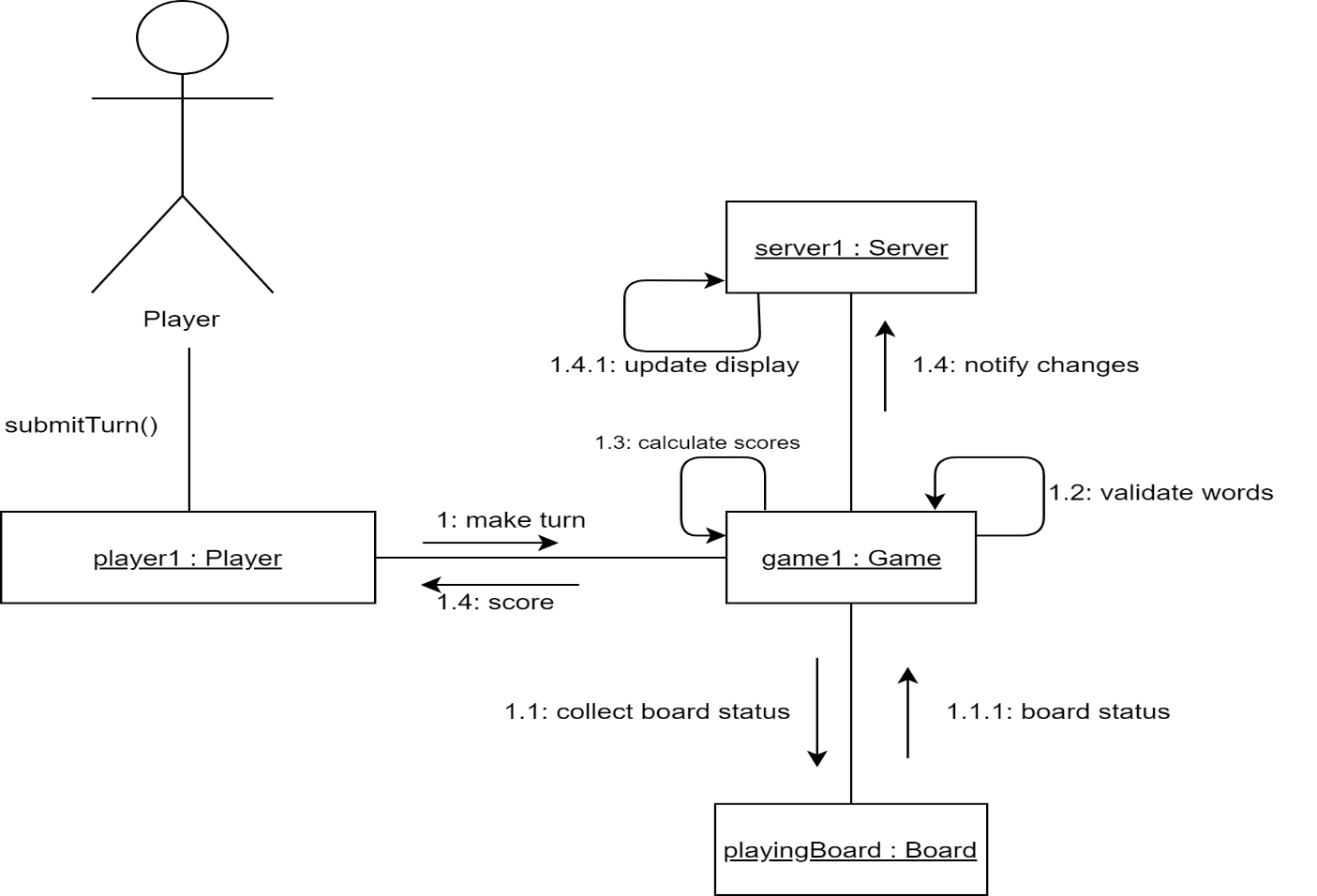
Join Game:

:

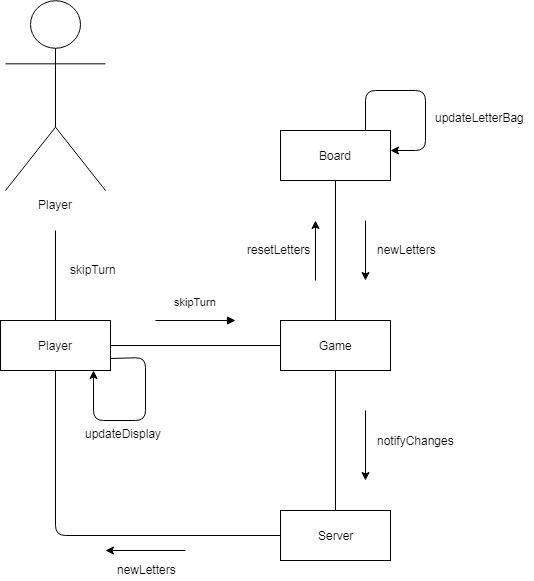
**Collaboration diagrams**

**Leave Game:**

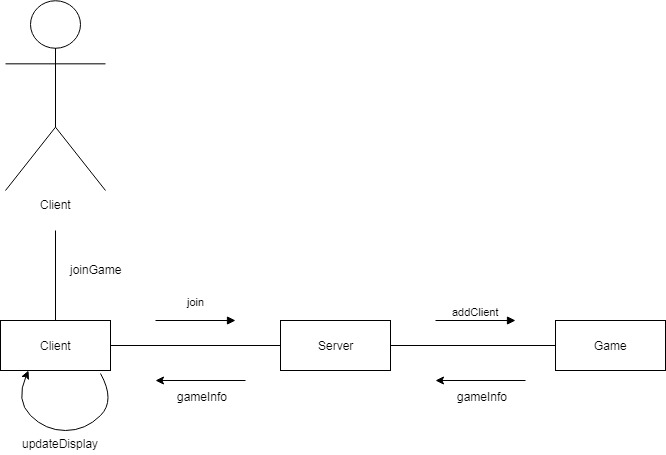
**Make Turn:**



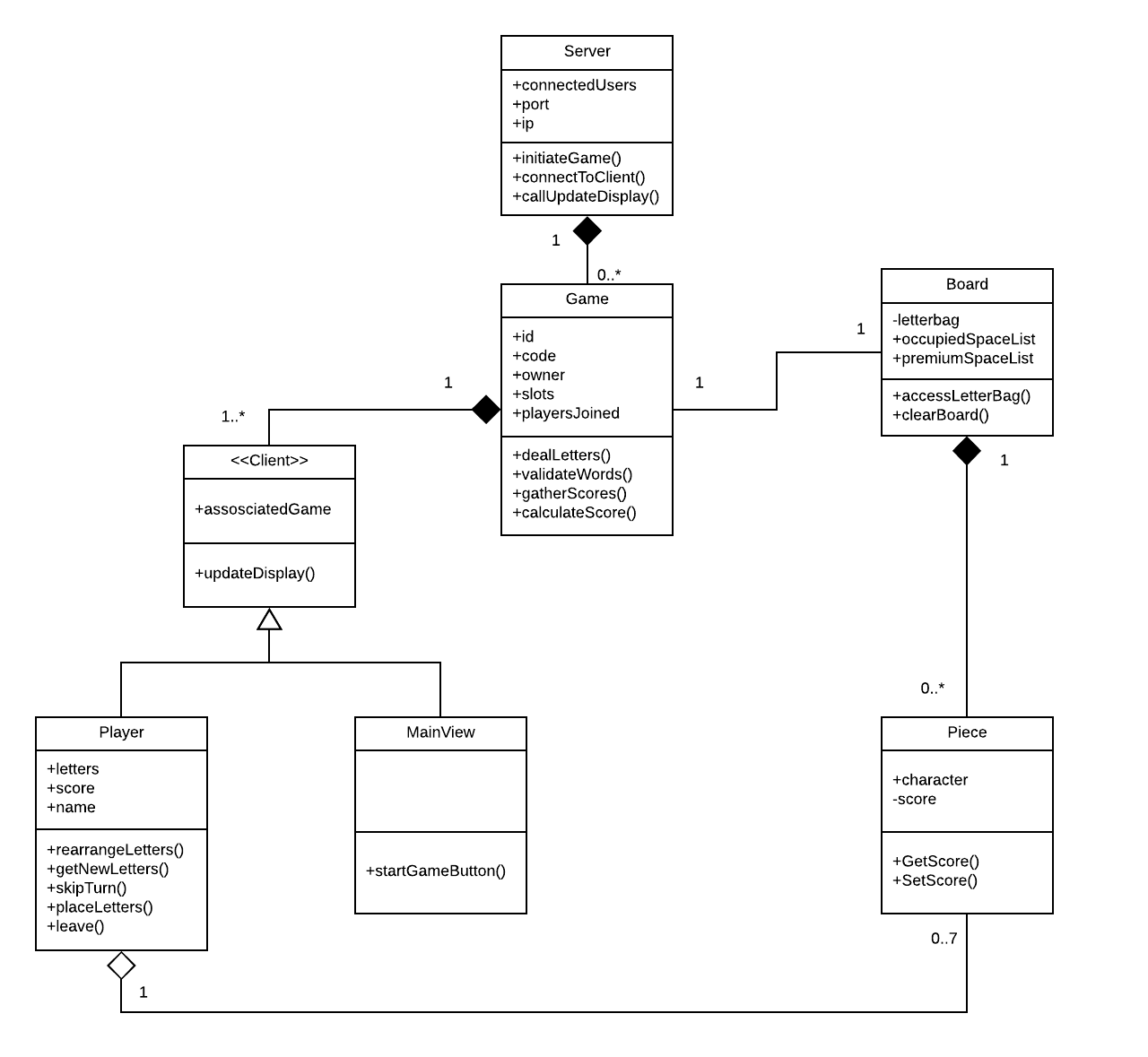
Skip Turn:



Join Game:



**More Refined Class Diagrams**



**Class Skeletons**

**Board.js:**

1. // Board class skeleton
3. **class** Board {
5. \_letterbag;
6. occupiedSpaceList;
7. premiumSpaceList;
9. constructor() {
10. **this**.\_letterbag = \_letterbag
11. **this**.occupiedSpaceList = occupiedSpaceList;
12. **this**.premiumSpaceList = premiumSpaceList;
13. }
15. accessLetterBag() {
16. // Function to retrieve a letter or put back a letter
17. }
19. clearBoard() {
20. // clear board of all letters.
21. }
23. }

**Client.js:**

1. // Client skeleton class
3. **class** Client {
5. \_associatedGame;
7. constructor(associatedGame) {
8. **this**.\_associatedGame = associatedGame;
9. }
11. updateDisplay() {
12. // Function to update the game display
13. }
15. }

**Mainview.js**

1. // MainView class Skeleton
3. **class** MainView {
5. startGameButton() {
6. // Start game from main screen
7. }
9. }

**Game.js**

1. // Game class skeleton
3. **class** Game {
5. constructor(id, code, owner, slots, playersJoined) {
6. **this**.id = id;
7. **this**.code = code;
8. **this**.owner = owner;
9. **this**.slots = slots;
10. **this**.playersJoined = playersJoined;
11. }
13. dealLetters() {
14. // Function which hands out random letters to each player
15. }
17. validateWords() {
18. // Function which checks if words are valid
19. }
21. gatherScores() {
22. // Function to collect scores of users
23. }
25. calculateScore(player) {
26. // Function to calculate a players score
27. }
29. }

**Piece.js**

1. // Piece class skeleton
3. **class** Piece {
5. \_score;
6. character;
8. constructor(character, score) {
9. **this**.character = character;
10. **this**.\_score = score;
11. }

14. get getScore() {
15. **return** **this**.\_score;
16. }
18. set setScore(value) {
19. **this**.\_score = value;
20. }
21. }

**Player.js**

1. // Player class skeleton
3. **class** Player {
5. constructor(letters, score, name) {
6. **this**.letters = letters;
7. **this**.score = score;
8. **this**.name = name;
9. }
11. rearrangeLetters() {
12. // Function to rearrange players letters
13. }
15. getNewLetters() {
16. // Get New letters from letter bag
17. }
19. skipTurn() {
20. // Skip players turn
21. }
23. placeLetters(coords) {
24. // PLayer letters on board
25. }
27. leave() {
28. // PLayer leaves game
29. }
31. }

**Server.js**

1. // Server class skeleton
3. **class** Server {
5. constructor(port, ip, connected\_users) {
6. **this**.port = port;
7. **this**.ip = ip;
8. **this**.connected\_users = connected\_users;
9. }
11. initiateGame() {
12. // Function will begin the game.
13. }
15. connectToClient() {
16. // Function which allows players to connect to each other
17. }
19. callUpdateDisplay() {
20. // Function which will update the screens display whenever there is change in game state
21. }
22. }

**Meeting Minutes**

**Minutes 31/10/2018**

**Absent:** Martin (Sick)

**What we did:**

* Planned what to have done

**What planned:**

* Research/Do Object Diagrams
* Research/Do Refined class diagrams
* Research/Do UI mock-ups
* Research/Do State Machines
* Research/Do Sequence Diagrams
* Research/Do Class Skeletons

**Minutes 5/1/2018**

**Absent:** Martin

**What we did:**

* Checked up on what done since last time (UI mock-up, object diagram)
* Planned what to do next

**What planned:**

* Research/Do Collaboration Diagrams.
* Research/Do Refined class diagrams.
* Research/Do Network prototype.
* Meet again Wednesday.

**Minutes 7/1/2018**

**Absent:** Nobody

**What we did:**

* Caught up on what everyone did.
* Researched state diagrams.

**What planned:**

* Research/Do state diagrams.
* Finish sequence and collaboration diagrams.

**Minutes 13/1/2018**

**Absent:** Nobody

**What we did:**

* Tidied what was done in final submission file.

**What planned:**

* Finish up on Thursday.

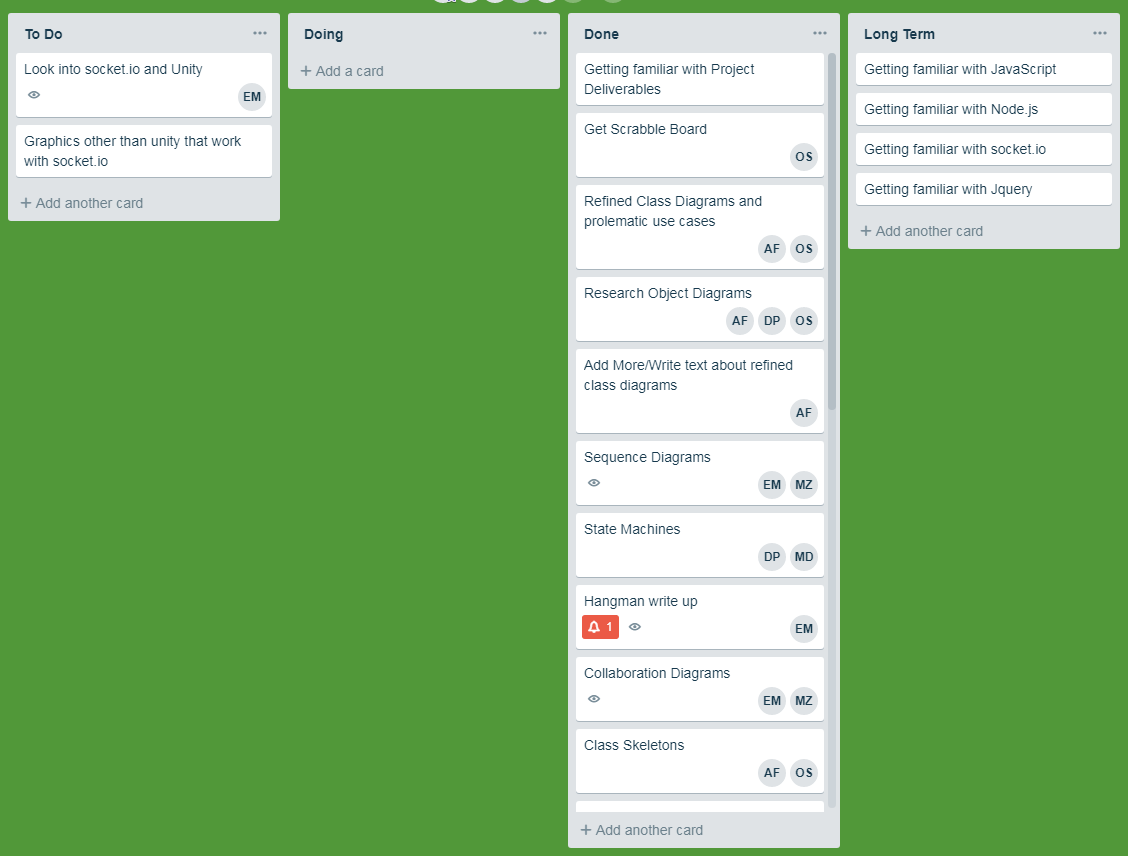
**Minutes 15/1/2018**

**Absent:** Mikey

**What we did:**

* Finished up deliverable
* Wrote short paragraph about Network features
* Wrote short paragraph about lack of need for updated object diagram.

**Once again we used Trello to track tasks:**



**We also used gitlab to track the files we worked on:**

