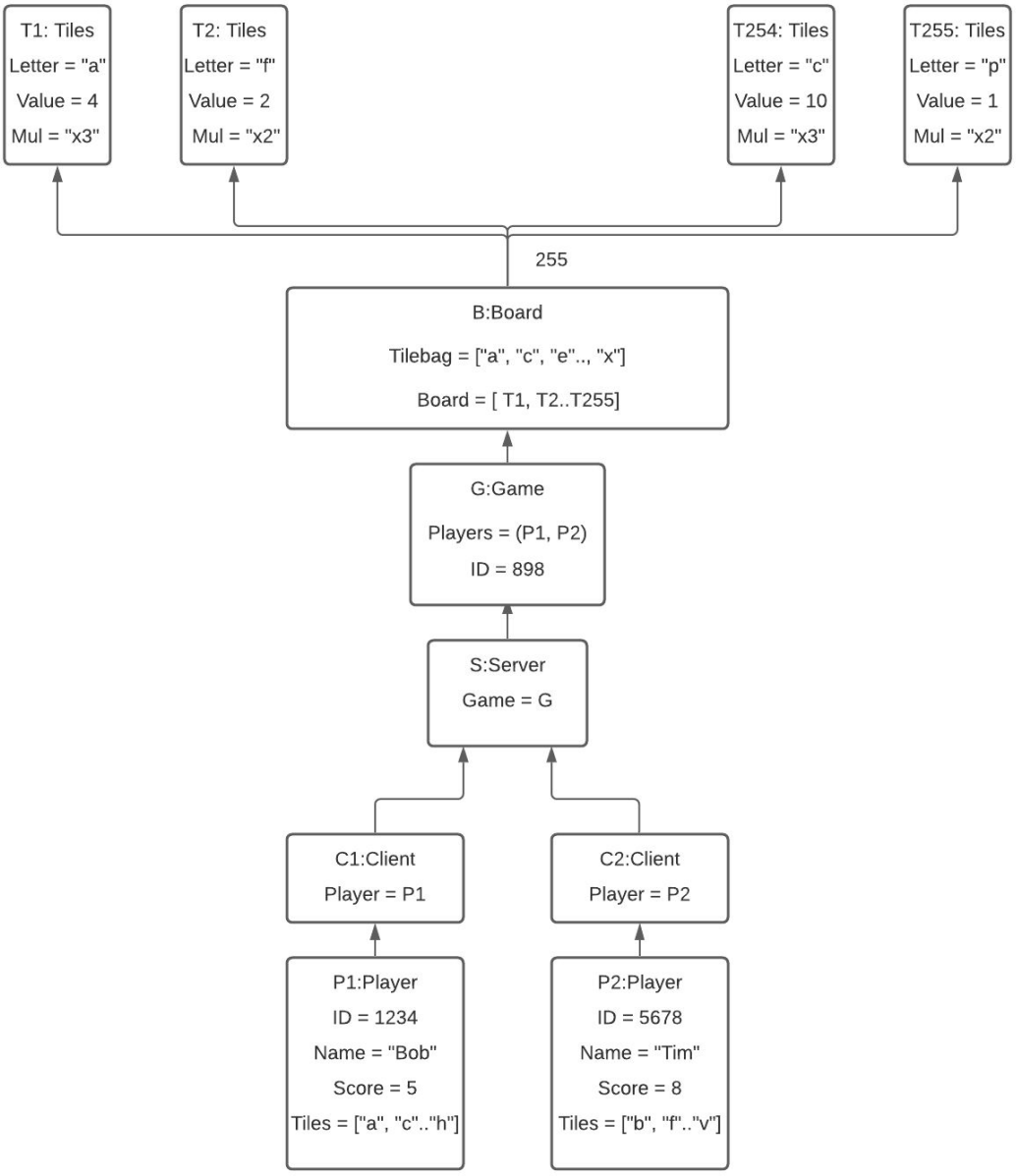
CA314 MyScrabble Product & Class Design

Group 1

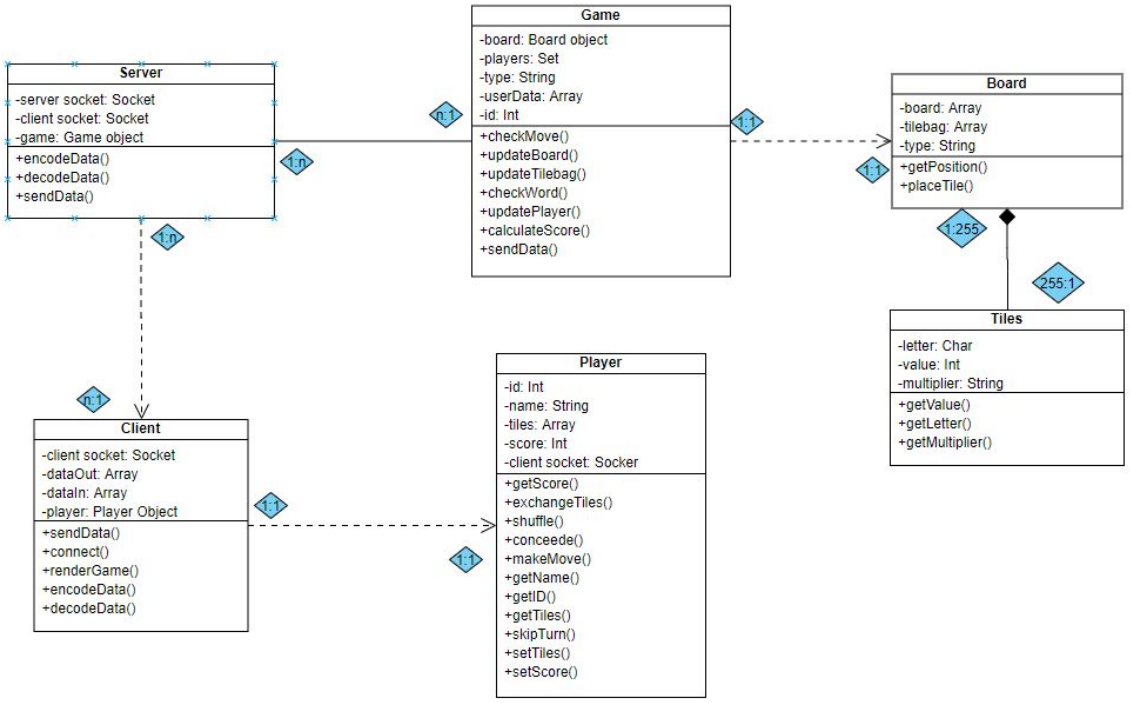
Table of Contents

Object Diagrams

Example of objects interacting during a game



Refined Class Diagrams



User Interface Mock-ups

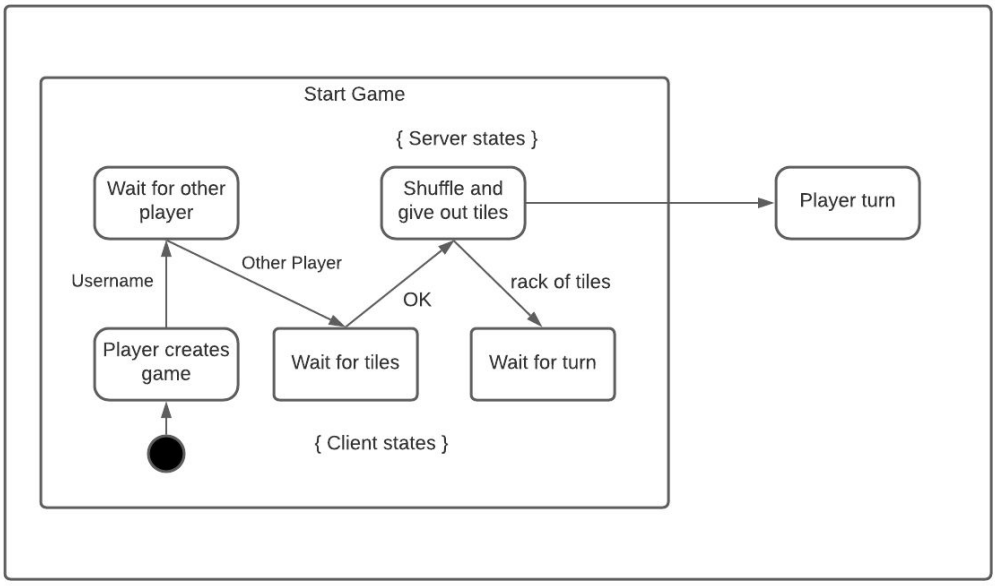
(1) ***Image representing the UI mockup developed for the main page of the site.***

(2) ***Image representing the UI mockup of the game in progress.***

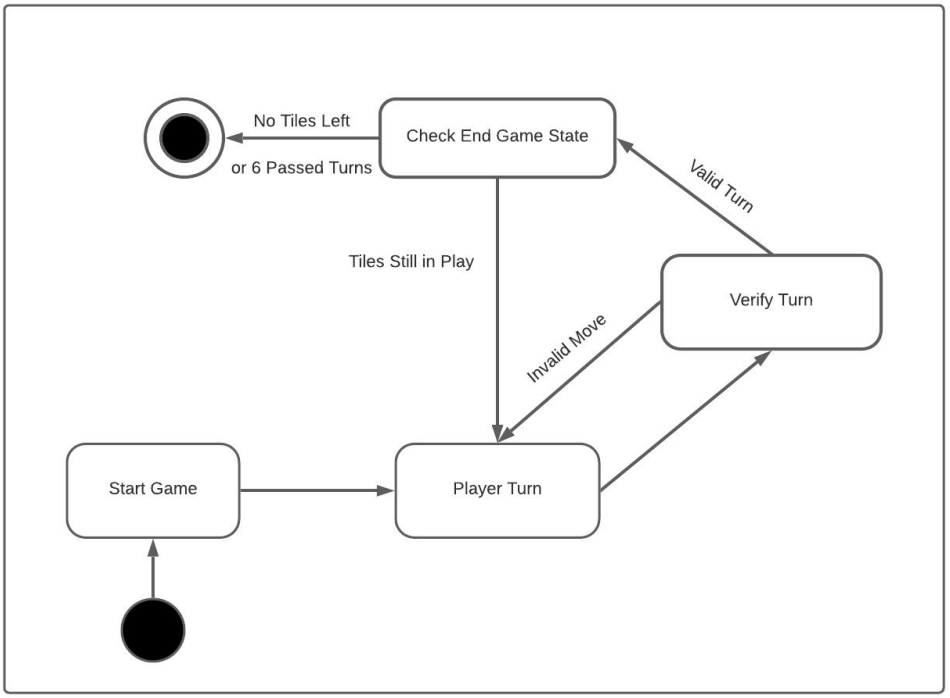
(3) *Image representing the UI mockup developed for join game state.*

State Machines

Start Game State Machine

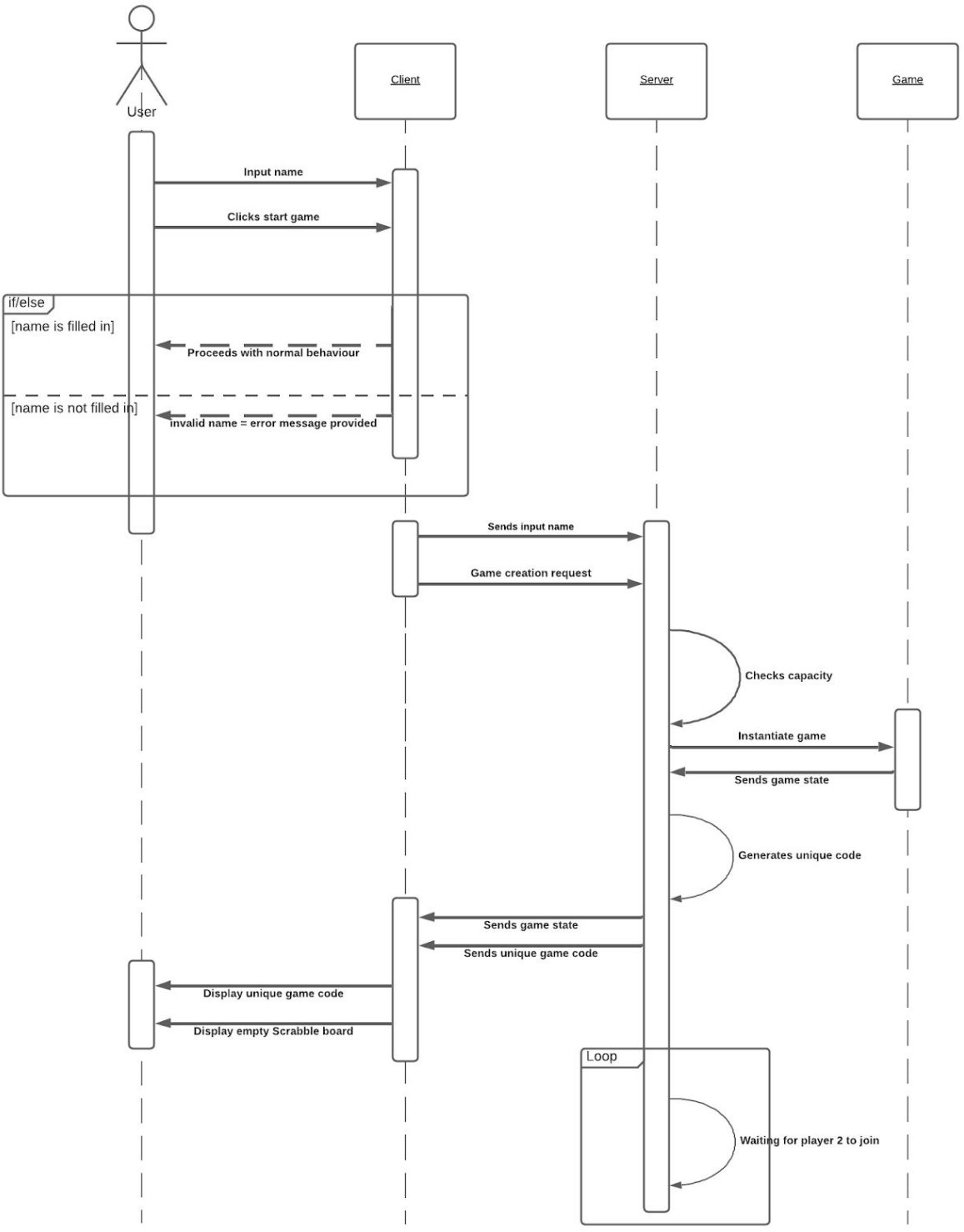


Program Overview State Diagram

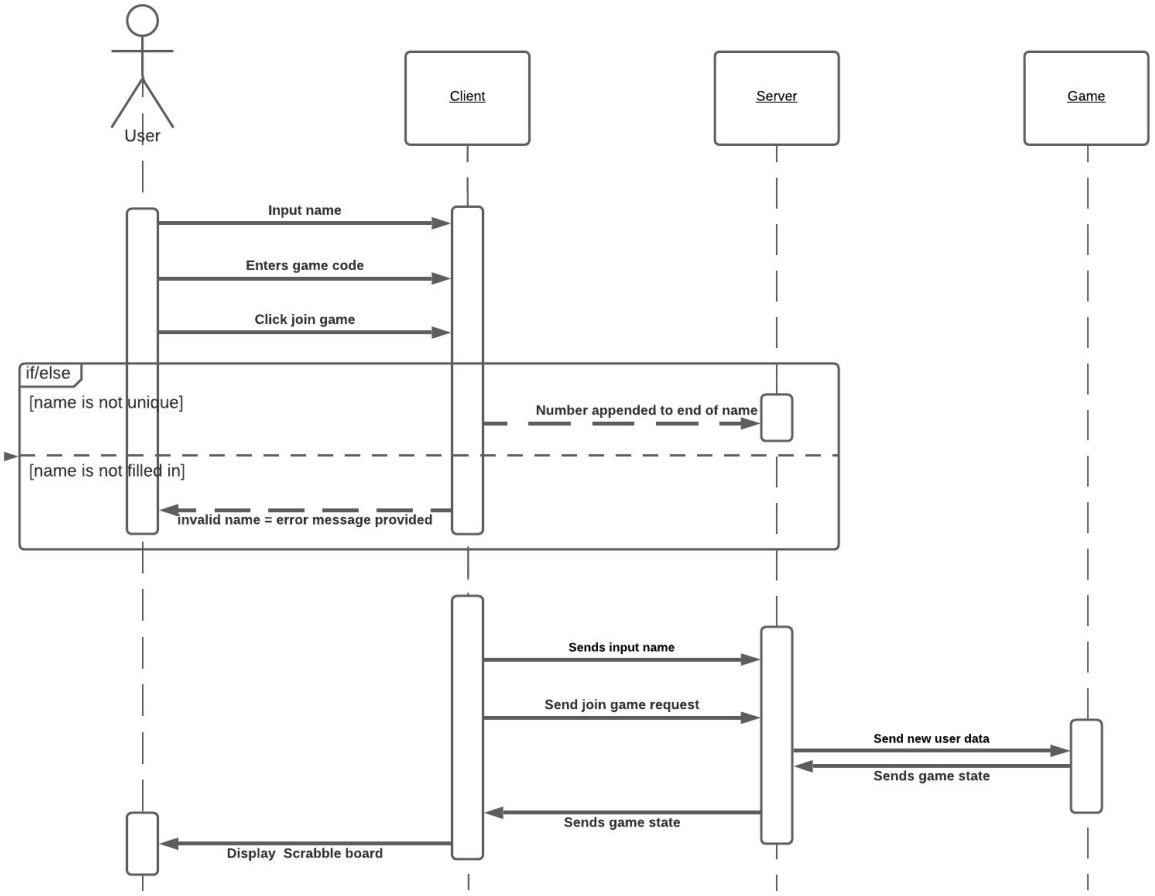


Sequence Diagrams

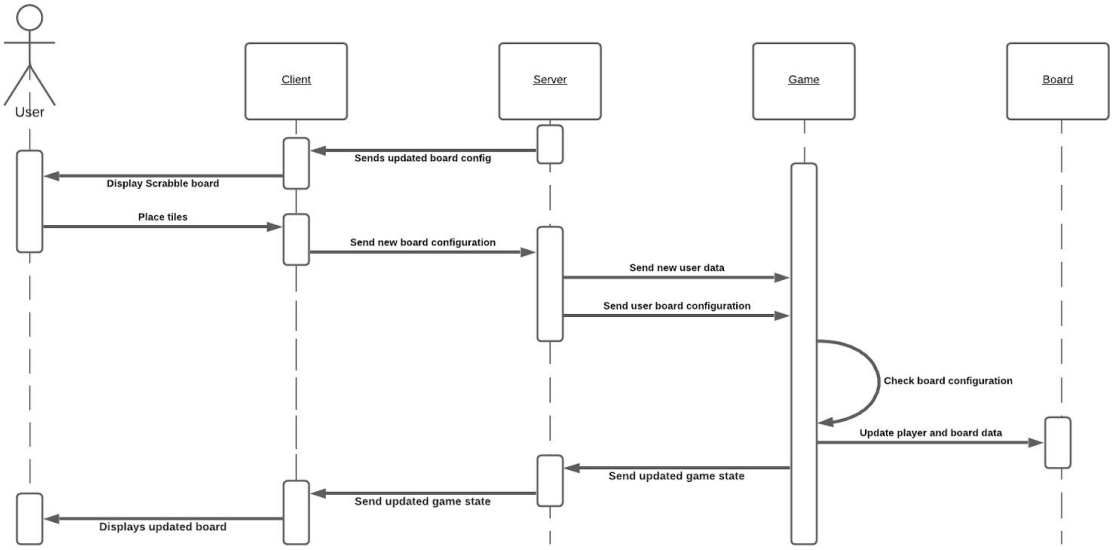
1. Create Game Use Case



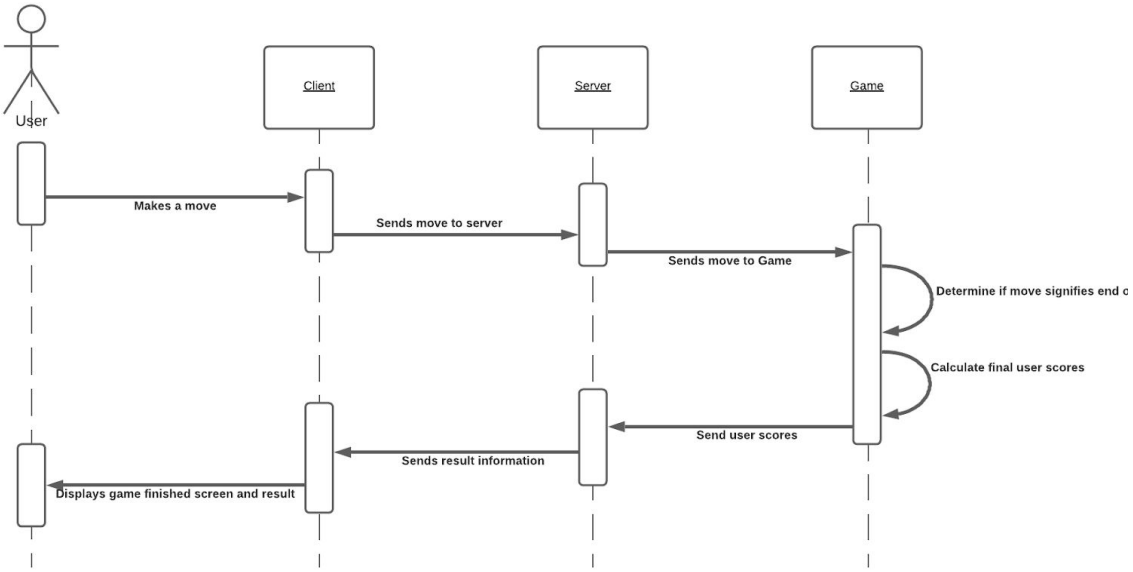
2. Join Game Use Case



3. Make Move Use Case

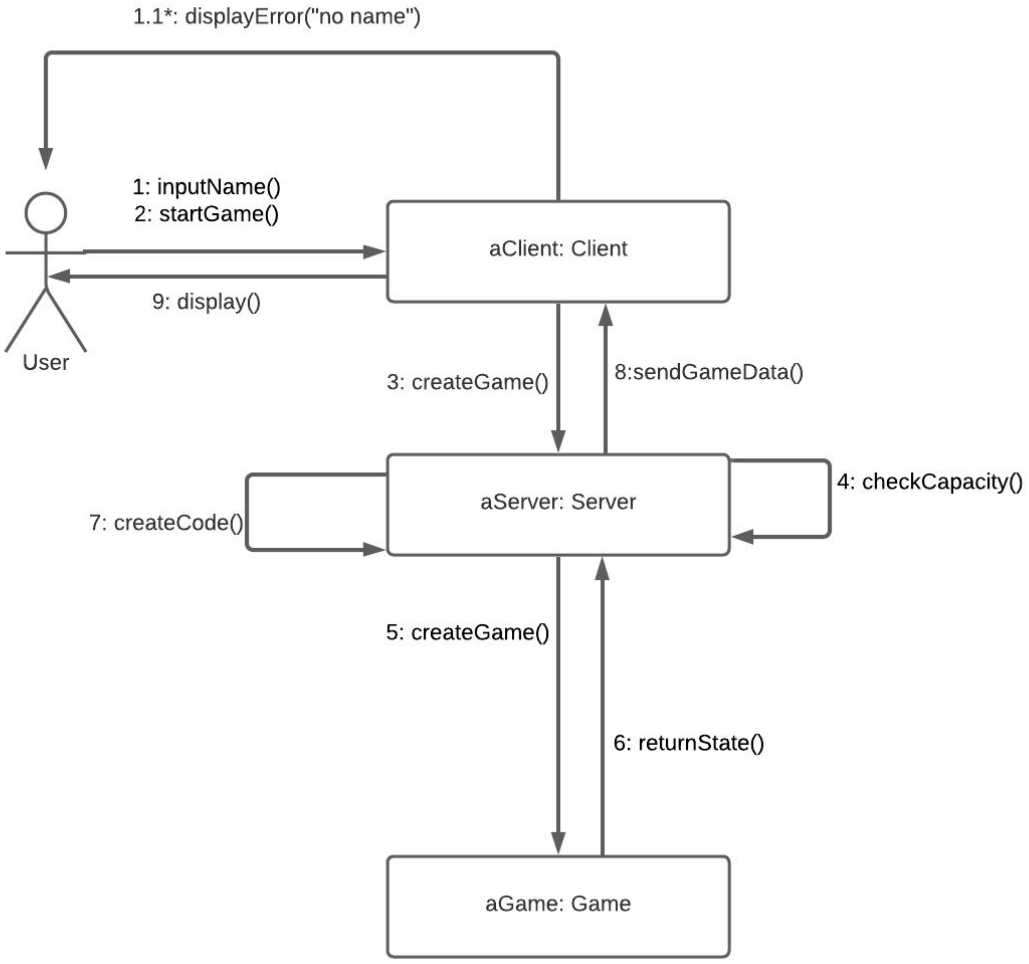


4. End Game Use Case

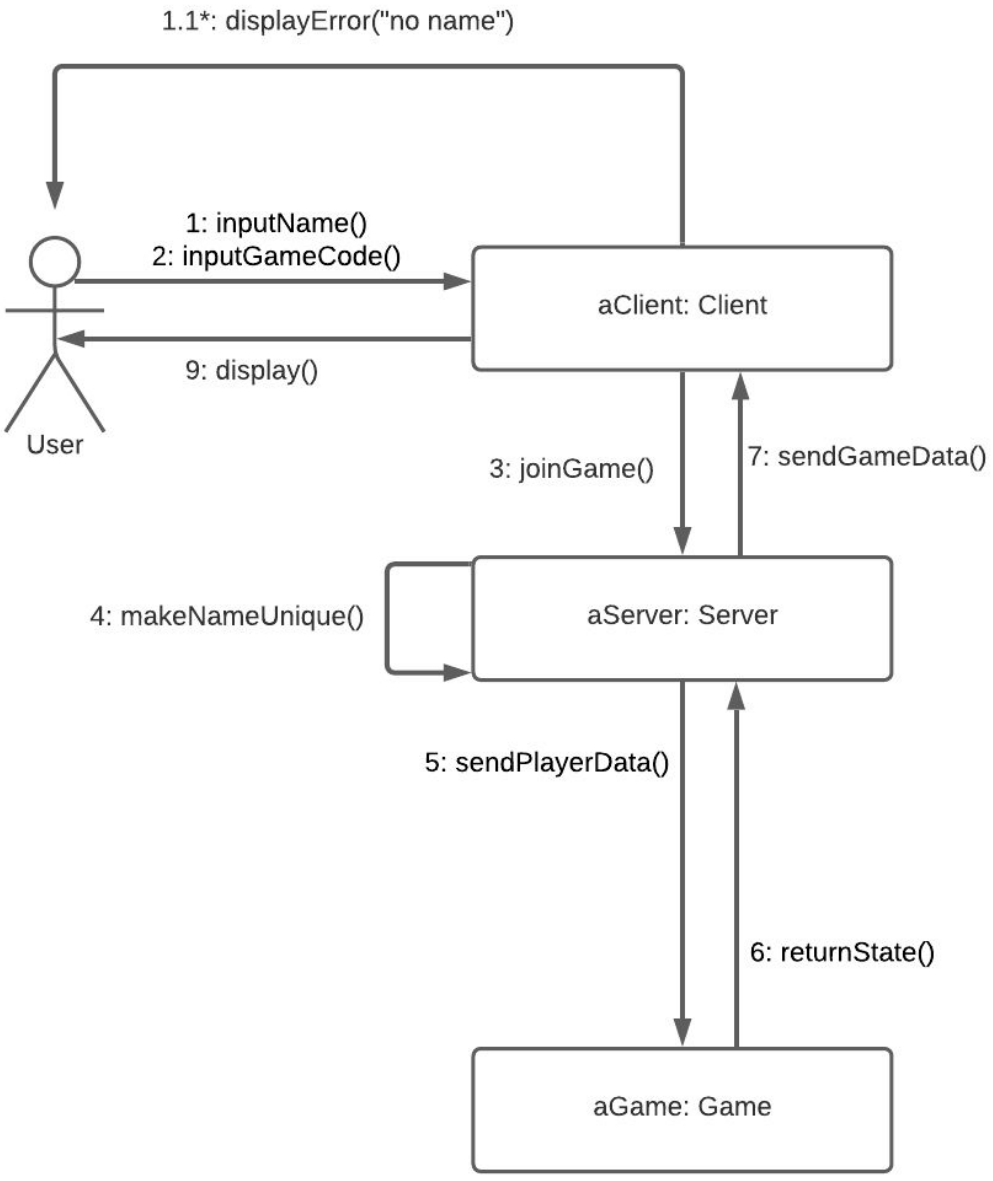


Collaboration Diagrams

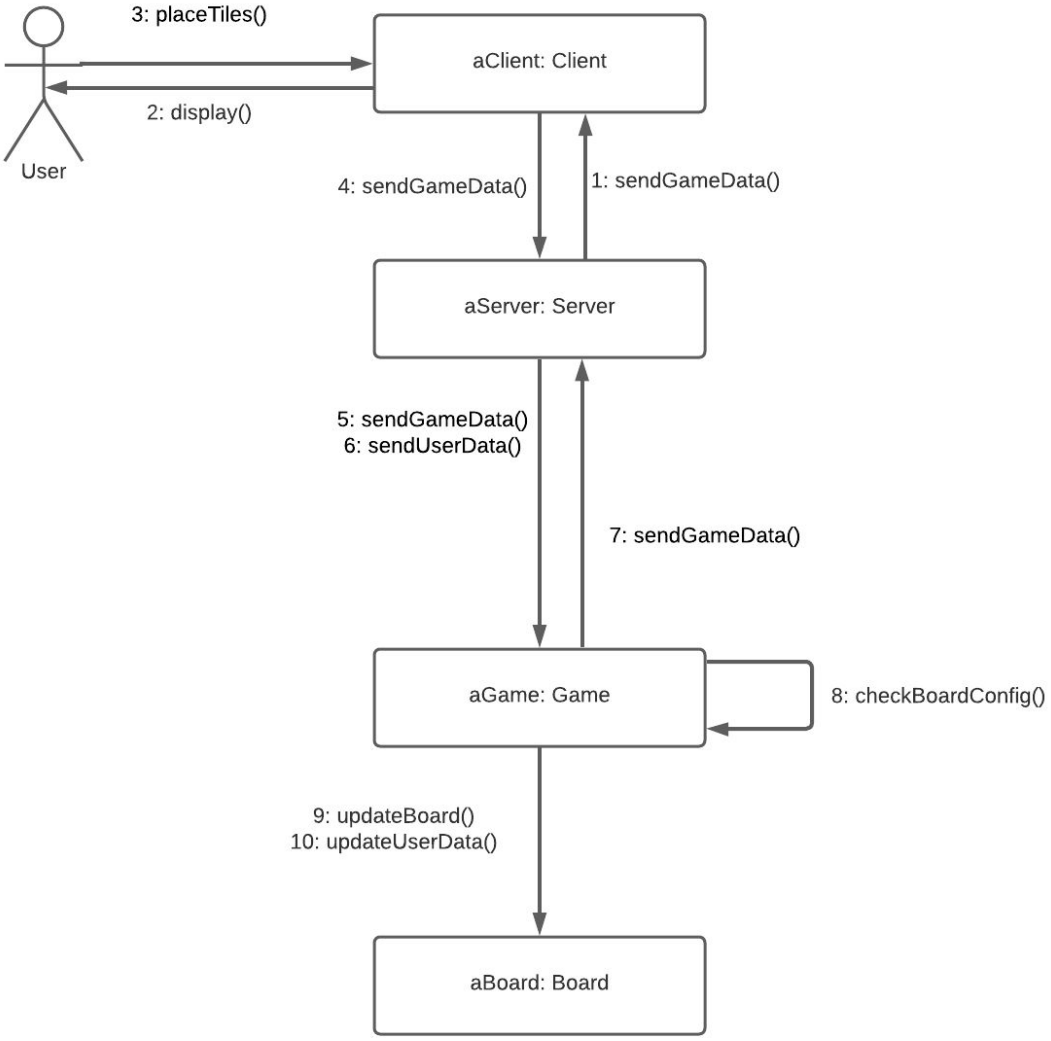
1. Create Game Use Case



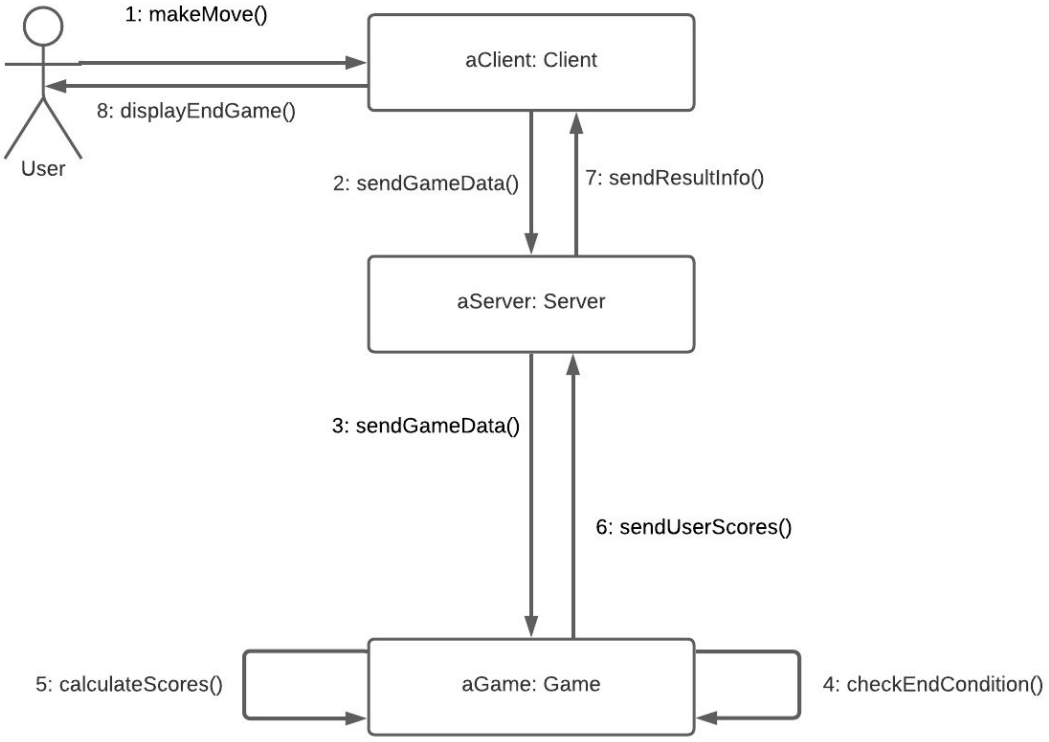
2. Join Game Use Case



3. Make Move Use Case

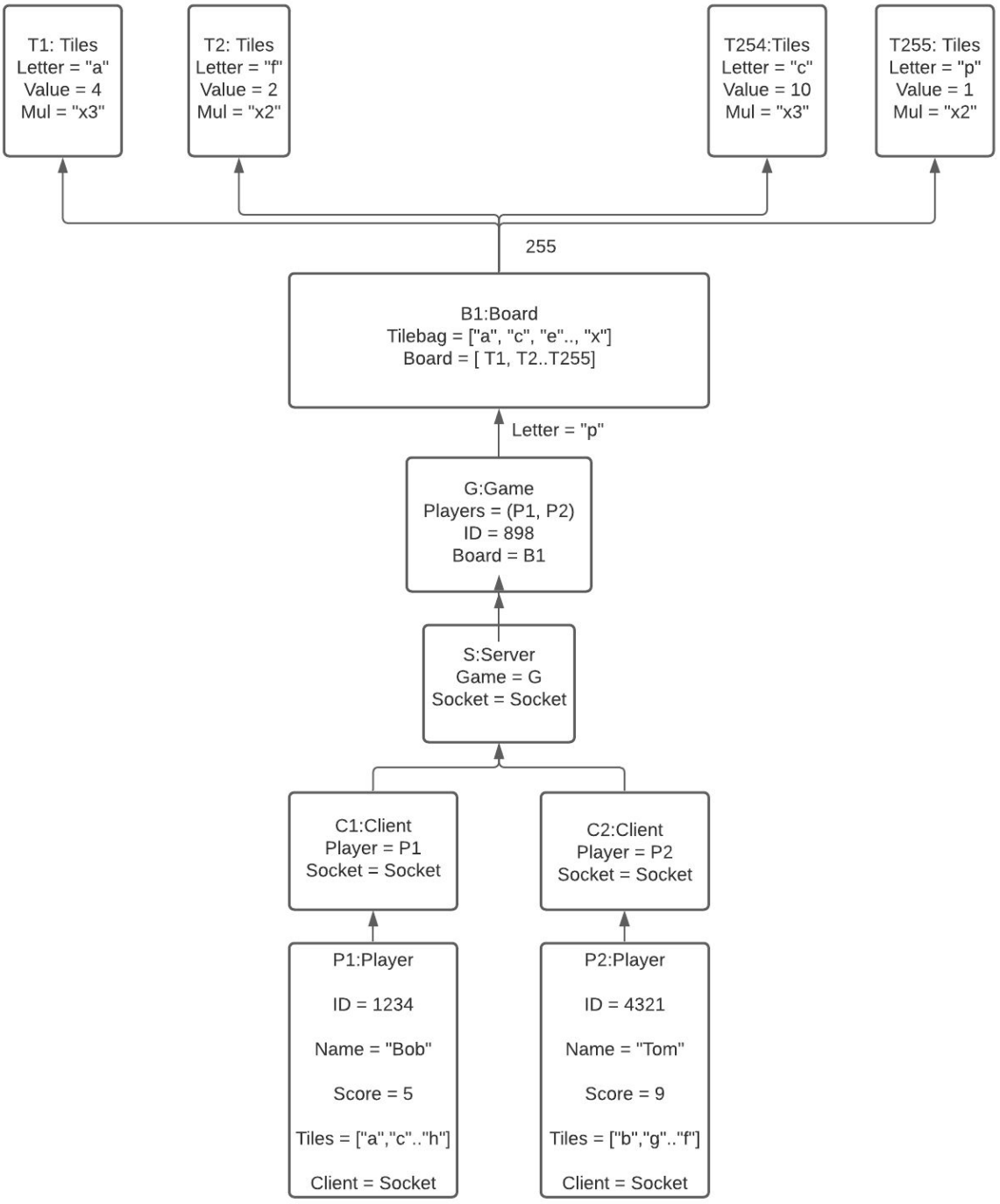


4. End Game Use Case

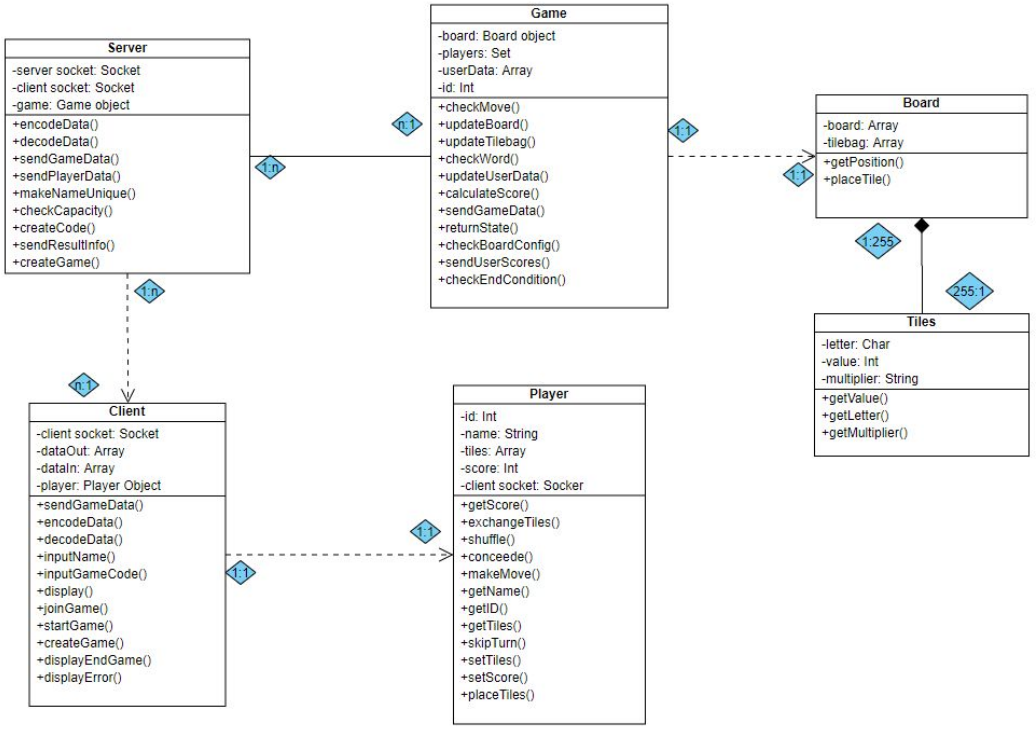


Revised Object Diagrams

Example of objects interacting during a game



Refined Class Diagrams



Class Skeletons

```
class Server:
      def __init__(self, socket, client_sockets, game):
      """
      Initializes an instance of the server class.
      :param socket: socket object for the server
      :param clients: a list of client objects
      :param game: a game object which handles the logic
      """
      self.socket = socket
      self.clients = clients
      self.game = game

      def encode(self, data):
      """
      :param data: Encoding data to be sent via
      socket to either a Client or Game object.
      """
      return encoded_data

      def decode(self, data):
      """
      :param data: Converts data into the correct
      format/structure to be parsed and then utilised.
      """
      return decoded_data
```

```
    def send_game_data(self, data, target):
    """
    Sends updated and encoded game state data to the client,'
    packaged/structured correctly for being
    interpreted by the Client.
    :param data: JSON object
    :param target: Client socket object
    """
    return # HTTP JSON data i.e status codes for whether the data was
sent successfully

    def send_player_data(self, data, target):
    """
    Sends data to the game, packaged/structured
    correctly for being interpreted by the recipient class.
    :param data: JSON Object
    :param target: Game instance for that Player.
    """
    return # does not return anything i.e Void

    def make_name_unique(self, name):
    """
    Takes an entered Player username that already exists
    in the target game, and makes it unique (by appending
    digits).
    :param name: String representation of username.
    """
    return unique_name

    def check_capacity(self):
    """
    Checks whether there is sufficient server
    capacity for a new client connection / game.
    """
    return # boolean

    def create_code(self):
    """
    Generates a unique ID code to represent
    the game and allow users to join.
    """
    return game_code

    def send_result_info(self):
    """
    Only used when game is ending, sends final results
    of game to client and also let's client know that
    the connection can be closed and the game has ended.
    """
    return # JSON object

    def create_game(self):
    """
    Initializes a game object when a game creation
    request is received.
    """
```

```
        return # void


class Client:
        def __init__(self, socket, data_out, data_in, player):
        """
        Initializes an instance of the client class.
        :param socket: socket object for the client
        :param data_out: JSON object
        :param data_in: JSON object
        :player: Player object
        """
        self.socket = socket
        self.data_out  = data_out
        self.data_in = data_in
        self.player = player

        def send_game_data(self, data, target):
        """
        Sends updated and encoded game state data to the server,'
        packaged/structured correctly for being
        interpreted by the Server.
        :param data: JSON object
        :param target: server socket object
        """
        return # void

        def encode(self, data):
        """
        :param data: Encoding data to be sent via
        socket to server.
        """
        return encoded_data

        def decode(self, data):
        """
        :param data: Converts data into the correct
        format/structure to be parsed and then utilised.
        """
        return decoded_data

        def input_name(self, name):
        """
        Updates player object's name
        :param name: player inputted name
        """
        return #void

        def input_game_code(self, code):
        """
        :param code: player inputted code
        """
        return code

        def display(self):
        """
```

```
        Render board object to users screen
        """
        return #void

        def join_game(self, code, target):
        """
        Passes player name and code to the server
        :param code: player inputted code
        :param target: object for code to be sent to
        """
        return #void

        def start_game(self):
        """
        Send game creation request to server
        """
        return #void

        def create_game(self, name):
        """
        Creates a game
        :param name: player name to be set
        """
        return #void


class Player:
        def __init__(self, id, name, tiles, score, client_socket):
        """
        Initializes an instance of the Player class.
        :param id: player unique id
        :param name: player unique name
        :param tiles: instantiates player tiles as empty list
        :param score: instantiates player score as 0
        :param client_socket: the player's client socket
        """
        self.id = id
        self.name = name
        self.tiles = []
        self.score = 0
        self.client_socket = client_socket

        def get_score(self):
        """
        Return player score
        """
        return self.score

        def exchange_tiles(self):
        """
        Send request to exchange tiles
        """
        return #void

        def shuffle(self):
        """
```

```
        Shuffles tiles on screen
        """
        return #void

        def make_move(self):
        """
        Send updated board to client
        """
        return #void

        def get_name(self):
        """
        Return player name
        """
        return self.name

        def get_ID(self):
        """
        Return id
        """
        return self.id

        def get_tiles(self):
        """
        Return player's tiles
        """
        return self.tiles

        def skip_turn(self):
        """
        send request to server to move to next player
        """
        return #void

        def set_tiles(self, tiles):
        """
        Update player tiles
        :param tiles: new tile list to be switched
        """
        return #void

        def set_score(self, score):
        """
        Update player score
        :param score: new player score
        """
        return #void

        def place_tiles(self):
        """
        Update game board
        """
        return #void

class Game:
        def __init__(self, board, players, user_data, ID):
```

```
        """
        Initializes an instance of the Game class.
        :param board: board object
        :param players: set of player objects
        :param user_data: array of user data
        :param ID: game unique id
        """
        self.board = board
        self.players = players
        self.user_data = user_data
        self.ID = ID

        def check_move(self):
        """
        Check board configuration for illegal move
        """
        return # bool

        def update_board(self):
        """
        Update board object with new configuration
        """
        return # void

        def update_tilebag(self):
        """
        Update board tile bag
        """
        return #void

        def check_word(self, word):
        """
        Check the dictionary if the word is legal
        :param word: word made by player
        """
        return #void

        def update_user_data(self):
        """
        Update user data
        """
        return #void

        def calculate_score(self):
        """
        Calculate word score
        """
        return score

        def send_game_data(self):
        """
        Send updated game object to server
        """
        return #void

        def return_state(self):
```

```
        """
        Return state of game
        """
        return state

        def check_board_config(self, board):
        """
        Check if board config is valid
        :param board: board object
        """
        return  # bool

        def send_user_scores(self):
        """
        Send player's scores to server
        """
        return self.players

        def check_end_conditoin(self):
        """
        Checks if game should end
        """
        return # bool

class Board:
        def  __init__(self, board, tilebag, type):
        """
        Initializes an instance of the Board class.
        :param board: the matrix (python lists)
        representation of the board.
        :param tilebag: an array to hold a series of random
        Tile objects.
        :param type: sets the type of the current Board
        based on the game type.
        """
        self.board = board
        self.tilebag = tilebag
        self.type = type

        def get_position(self, pos):
        """
        Getter to return the Tile object at
        a particular position on the board.
        """
        return board[pos]

        def place_tile(self):
        """
        Place a new Tile "onto" the board.
        """
        return board

class Tile:
    def __init__(self, letter, value, multiplier):
        """
        Initializes an instance of the Tile class.
```

```
        :param letter: string/char representation of a letter (A-Z)
        :param value: an integer value to represent the value
        of the letter / tile.
        :param multiplier: sets the multiplier type of the tile.
        This is used for managing multiplier tiles on the board.
        """
        self.letter = letter
        self.value = value
        self.multiplier = multiplier

    def get_value(self):
        """
        Getter to return value attribute of
        the tile object.
        """
        return self.value

    def get_letter(self):
        """
        Getter to return letter attribute of
        the tile object.
        """
        return self.letter

    def get_multiplier(self):
        """
        Getter to return the multiplier
        attribute of the tile object.
        """
        return self.multiplier
```

Minutes/Notes of Team Meetings

All members were present during our meetings.

Meeting 01 - 03/11/2020

Sprint #1 Meeting

- First meeting was an overall collaborative review of the next section of the document and we began to work through the key points and items that needed to be addressed in Section 3.2.

- We assigned each team member their specific roles:
  - Object Diagrams -> Stefan
  - Refined Class Diagrams -> Maciej & James
  - UI Mock-ups -> Cian
  - State Machines -> Marius

Meeting 02 - 10/11/2020

Sprint #2 Meeting

- This meeting began with the group sharing their independent work set for them during the previous week.

- The team then worked on the Collaboration and Sequence Diagrams together using lucidchart.

- We then split up the work as follows:

    - James and Maciej will work together on the refined class and object diagrams.

    - Stefan, Marius and Cian designed the class skeletons.

Meeting 03 - 17/11/2020

Sprint #3 Meeting

- In this meeting, we reviewed each other's work that we had assigned and provided feedback.

- We concluded that some changes needed to be made and applied them. Following this, we stepped through the document with these additional sections completed and verified the quality throughout.

- Lastly, we finalized the document by refining the structure and appearance.