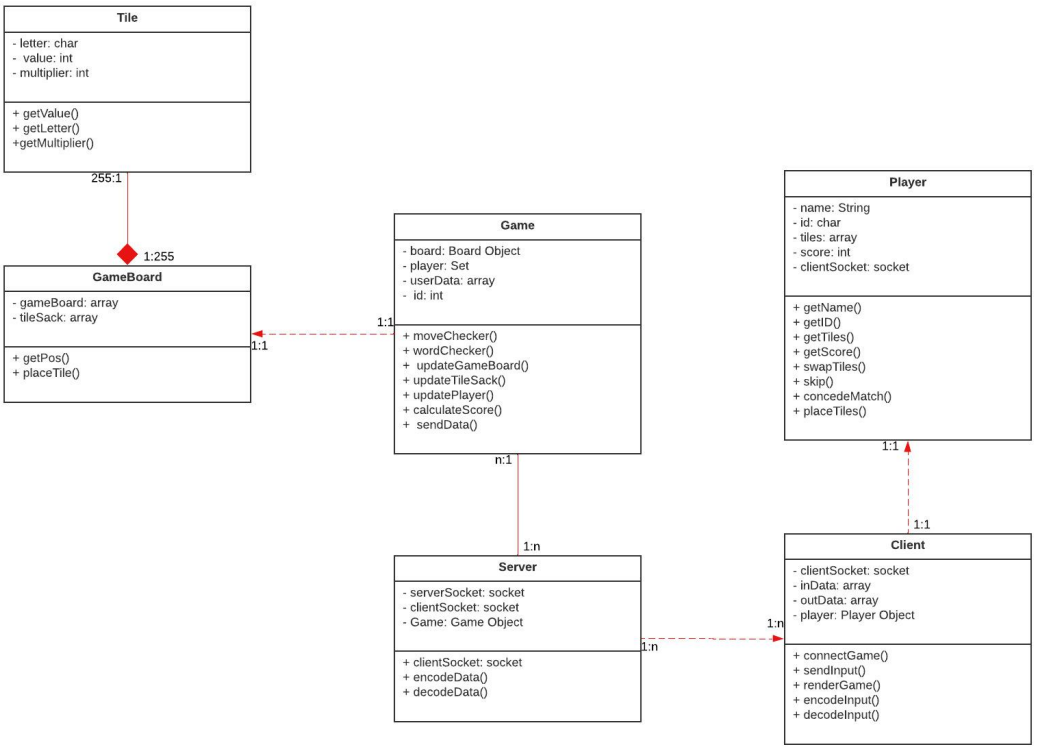
CA314 Assignment 2 - Scrabble Product and Class Design

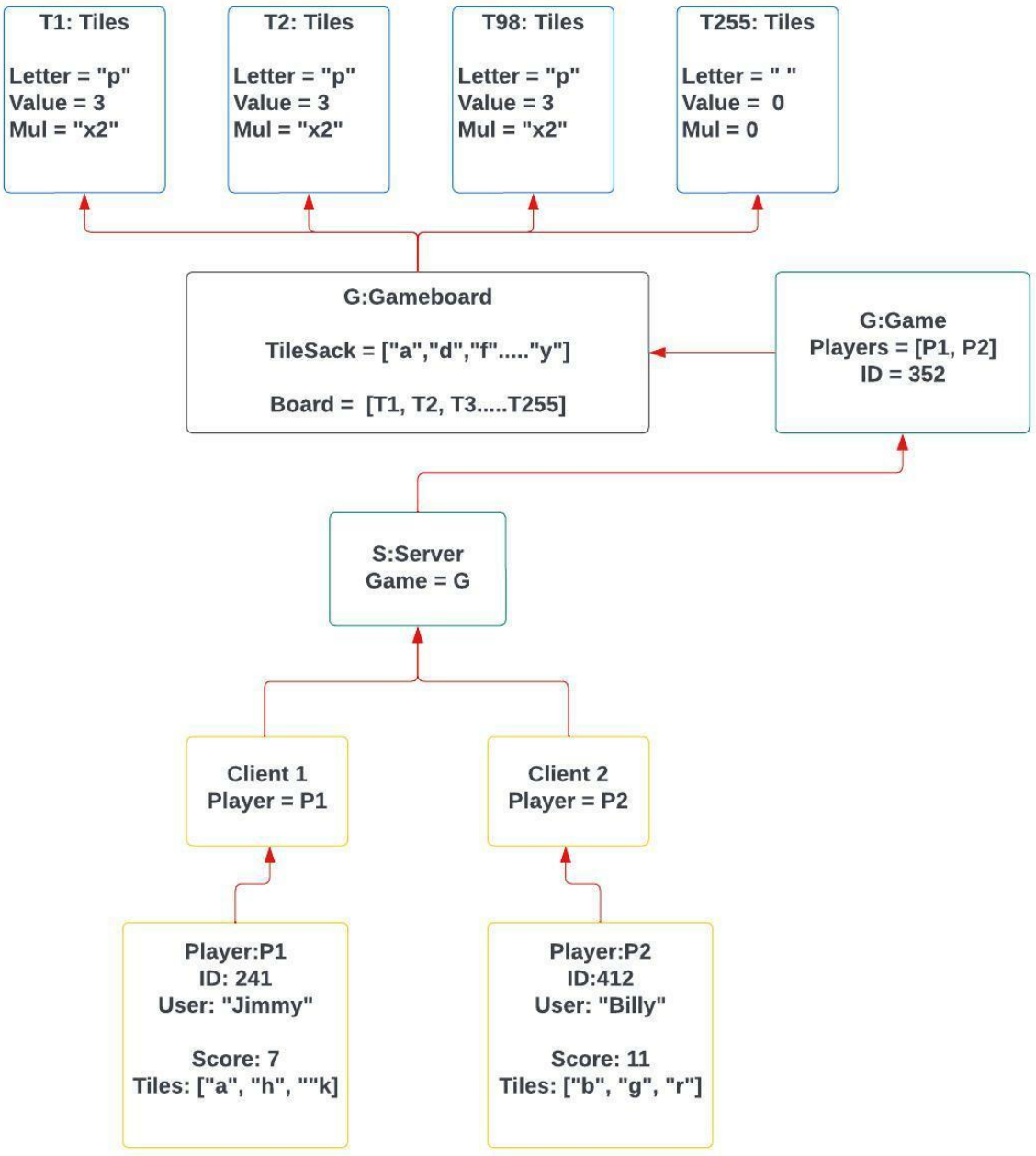
Group 5

Table of Contents

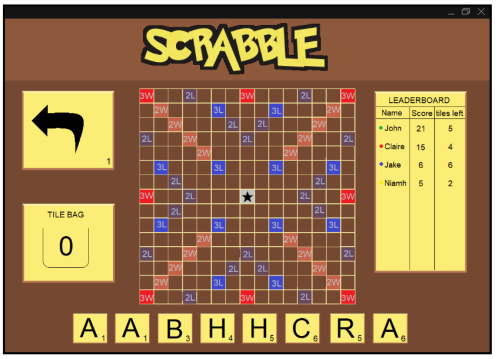
Refined Class Diagrams



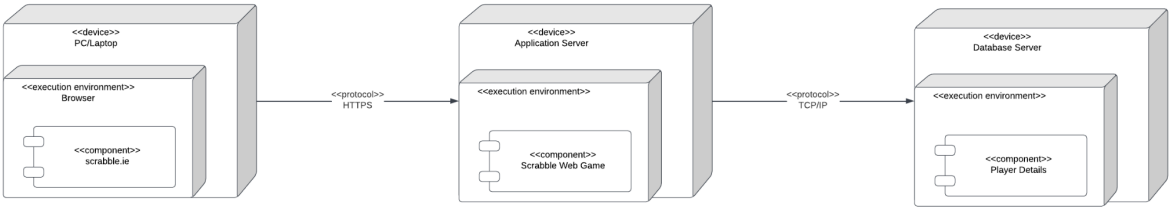
Object Diagrams



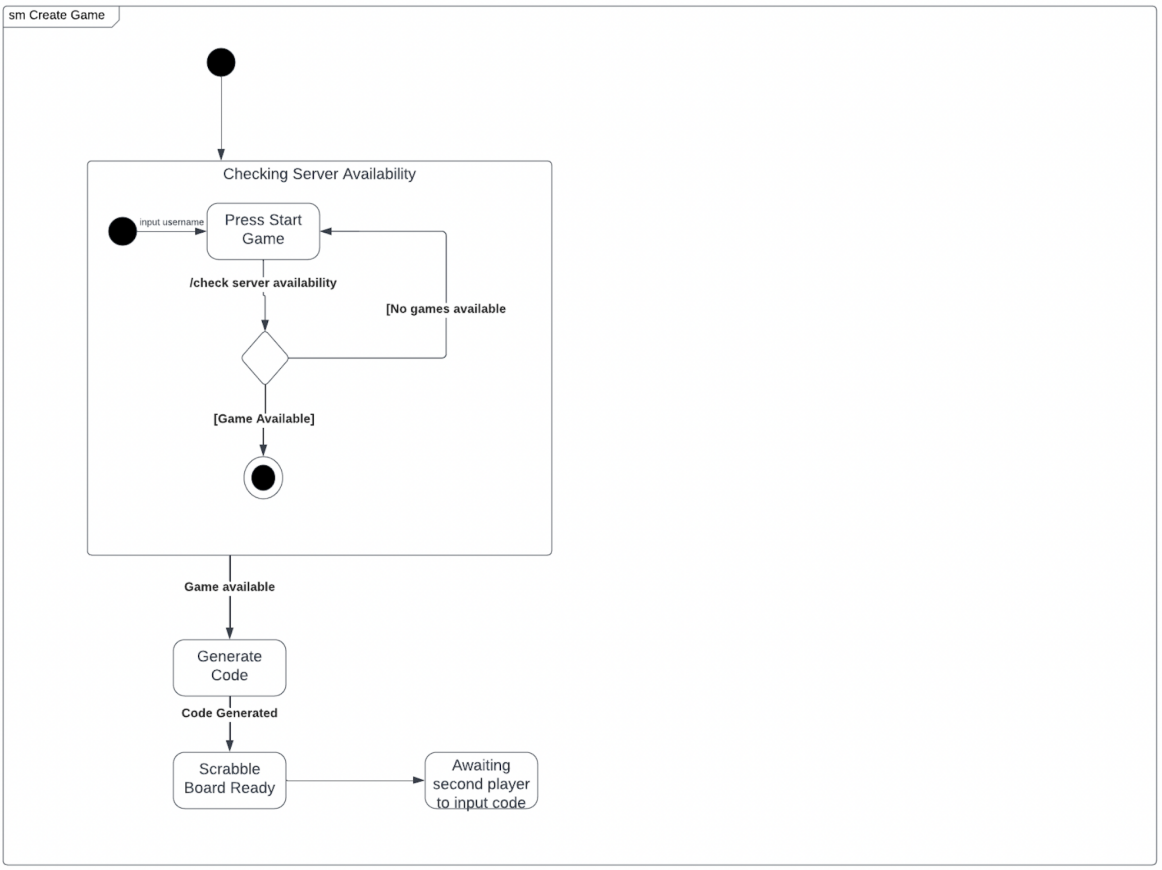
User Interface Mock-ups

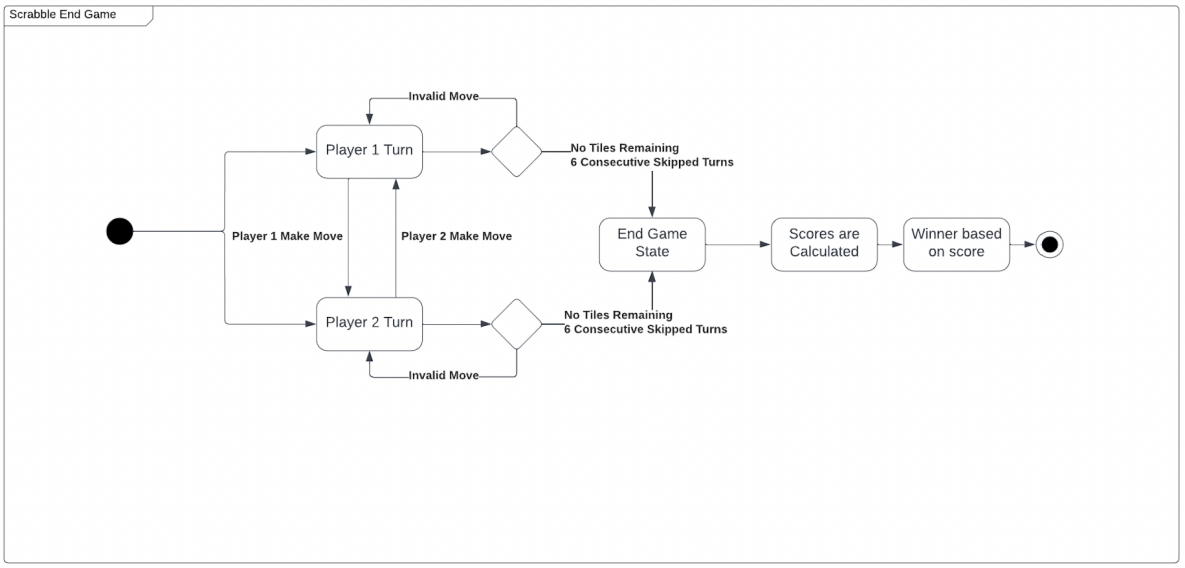


Network Ideas

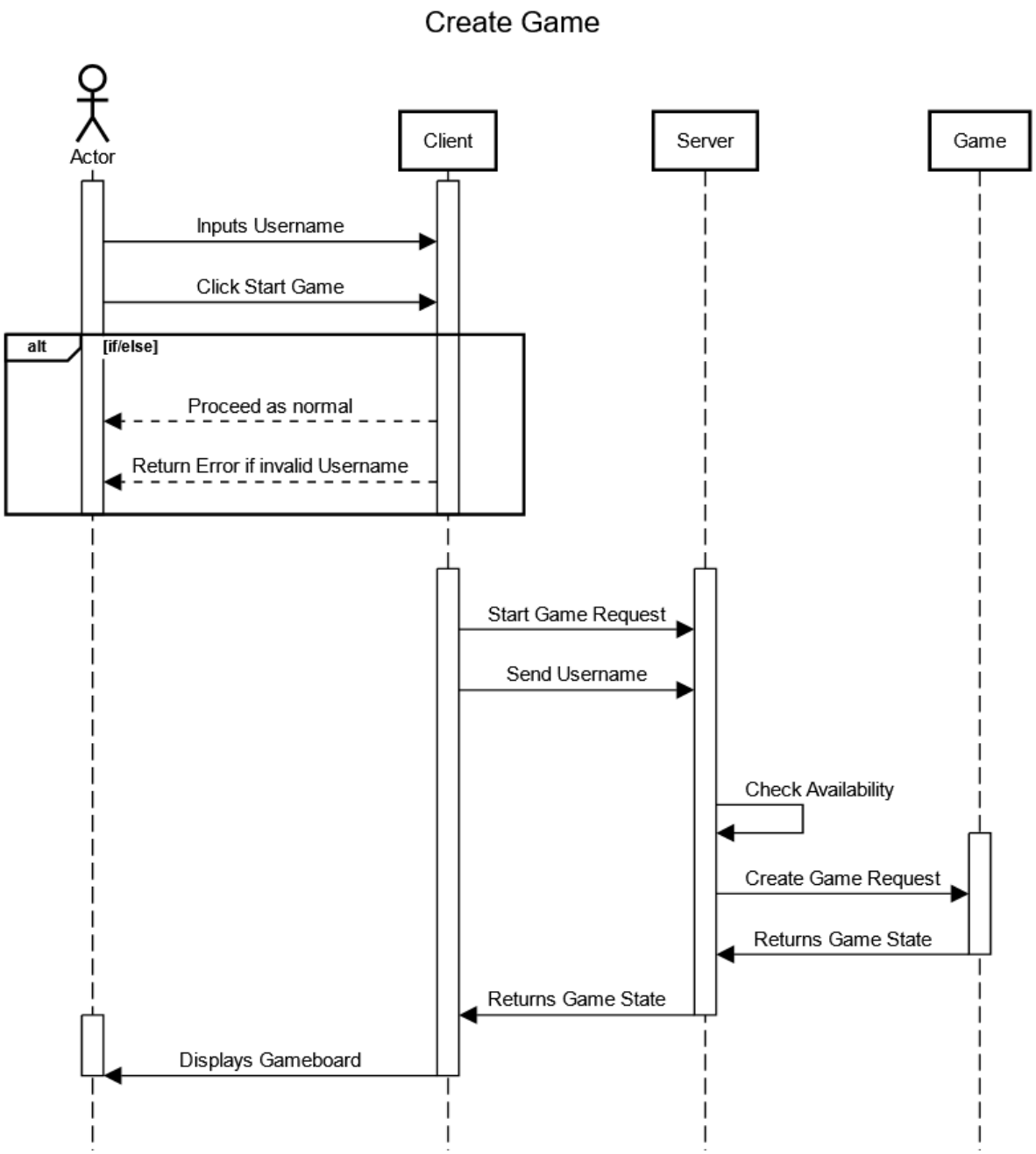


State Machines

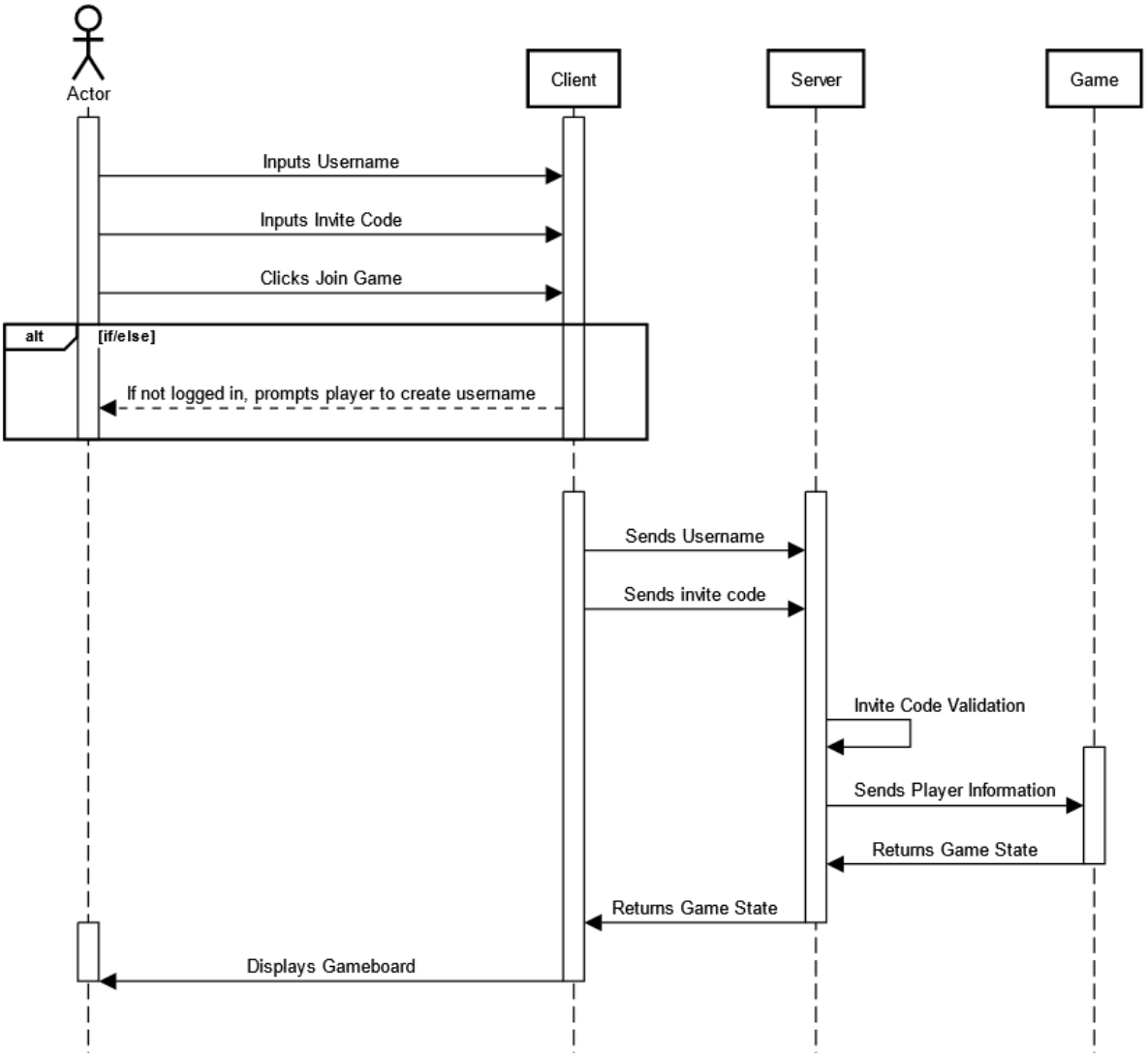




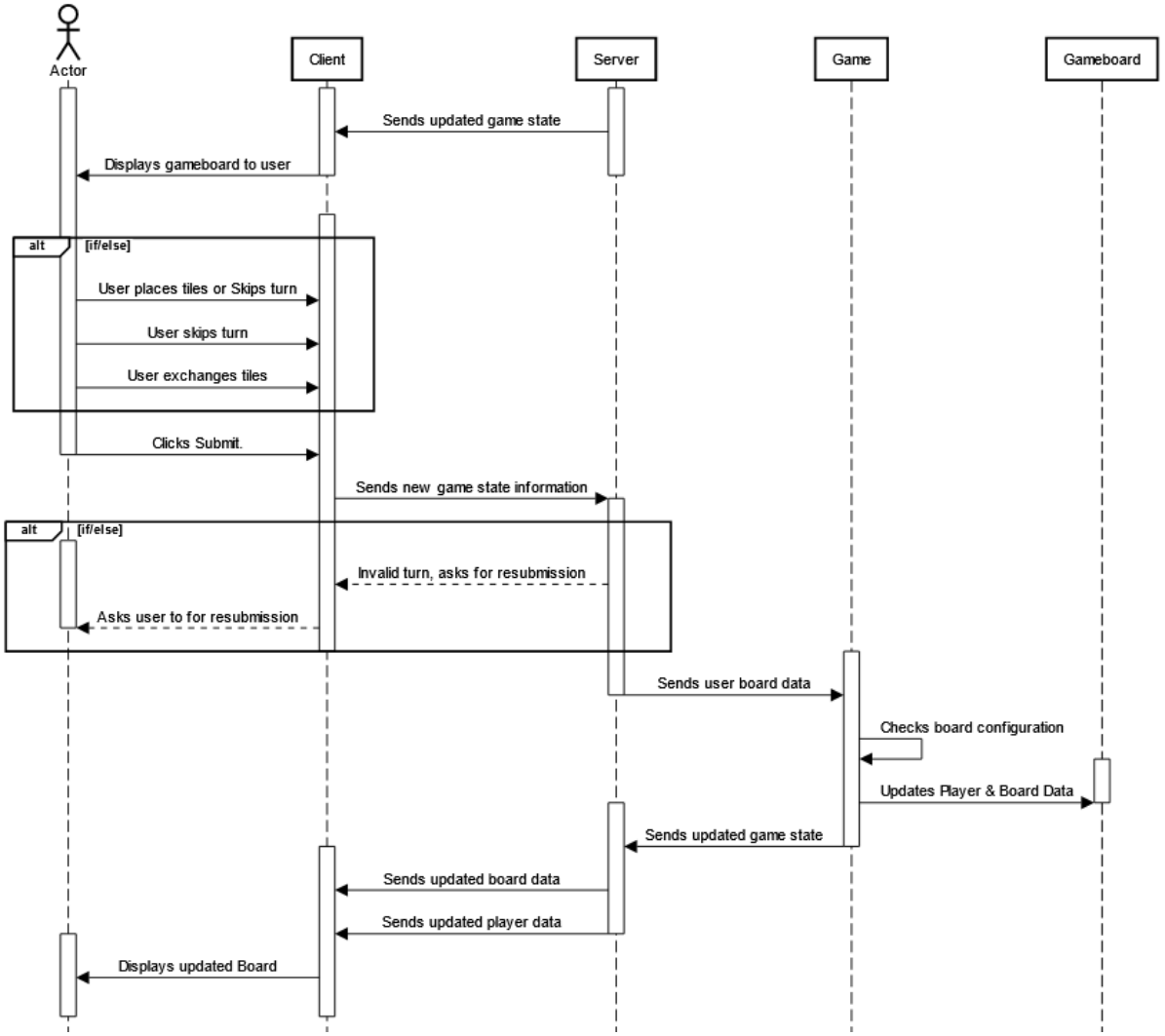
Sequence Diagrams



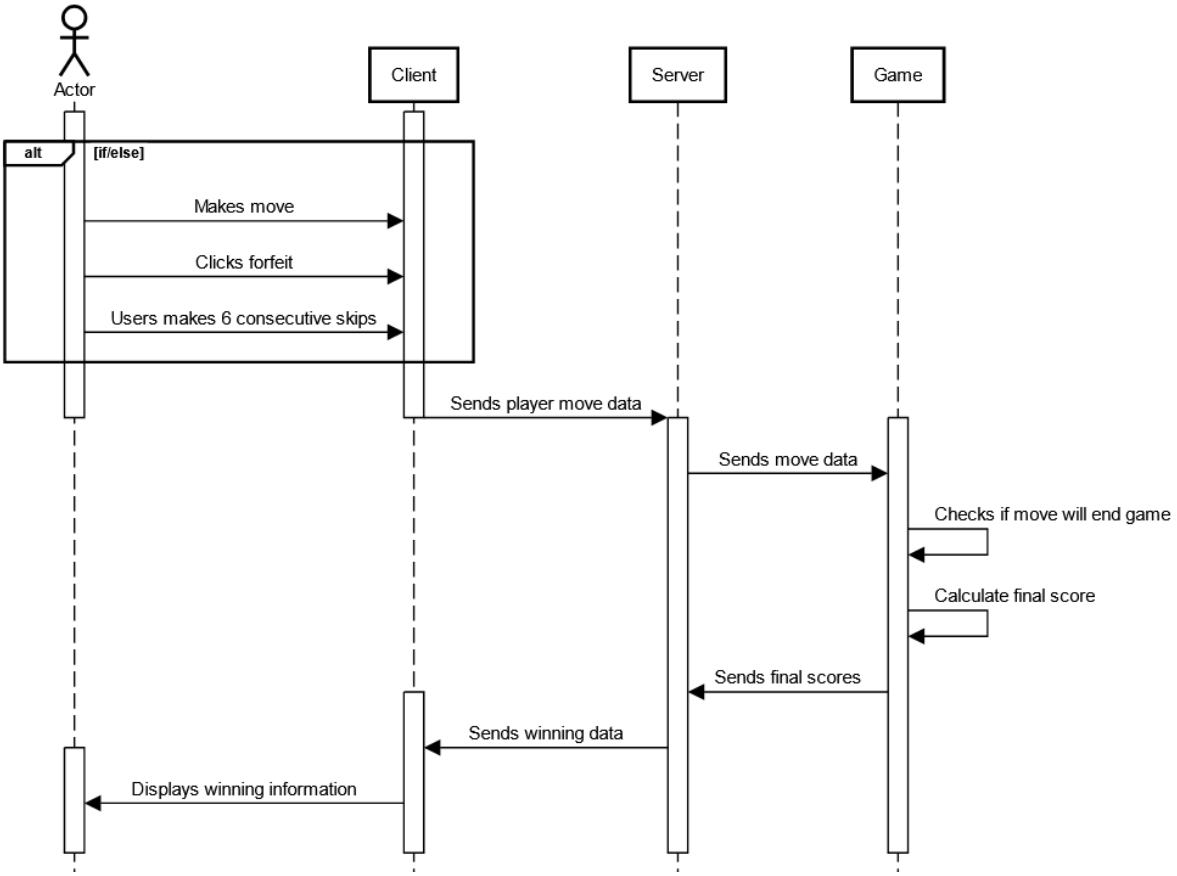
Join Game


Actor
Client
Server
Game
Inputs Username
Inputs Invite Code
Clicks Join Game
alt
[if/else]
If not logged in, prompts player to create username
Sends Username
Sends invite code
Invite Code Validation
Sends Player Information
Returns Game State
Returns Game State
Displays Gameboard

Make Move

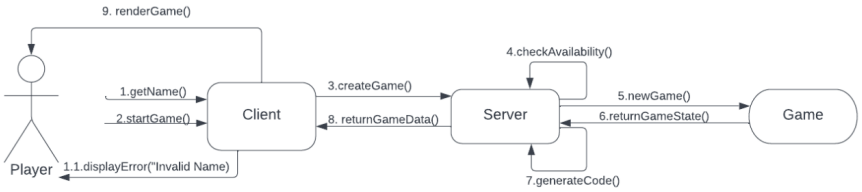

Actor
Client
Server
Game
Gameboard
Sends updated game state
Displays gameboard to user
alt
[if/else]
User places tiles or Skips turn
User skips turn
User exchanges tiles
Clicks Submit.
Sends new game state information
alt
[if/else]
Invalid turn, asks for resubmission
Asks user to for resubmission
Sends user board data
Checks board configuration
Updates Player & Board Data
Sends updated game state
Sends updated board data
Sends updated player data
Displays updated Board

Game End

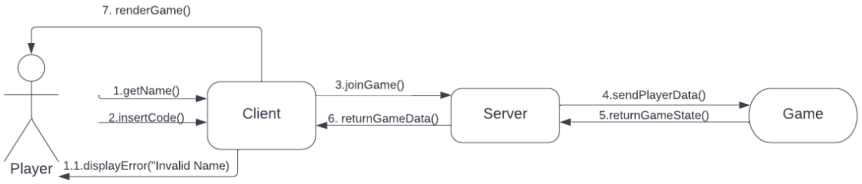

Actor
Client
Server
Game
alt
[if/else]
Makes move
Clicks forfeit
Users makes 6 consecutive skips
Sends player move data
Sends move data
Checks if move will end game
Calculate final score
Sends final scores
Sends winning data
Displays winning information

Collaboration Diagrams

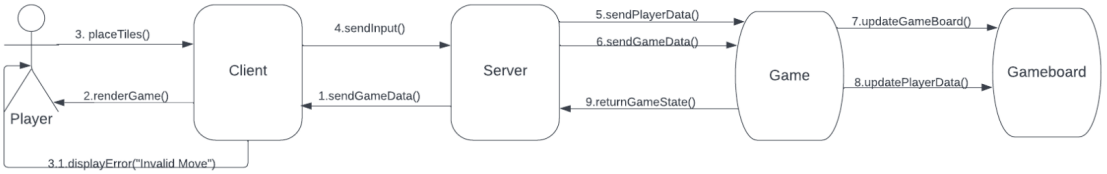
Create Game



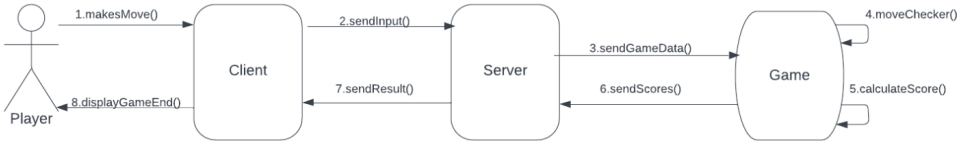
Join Game



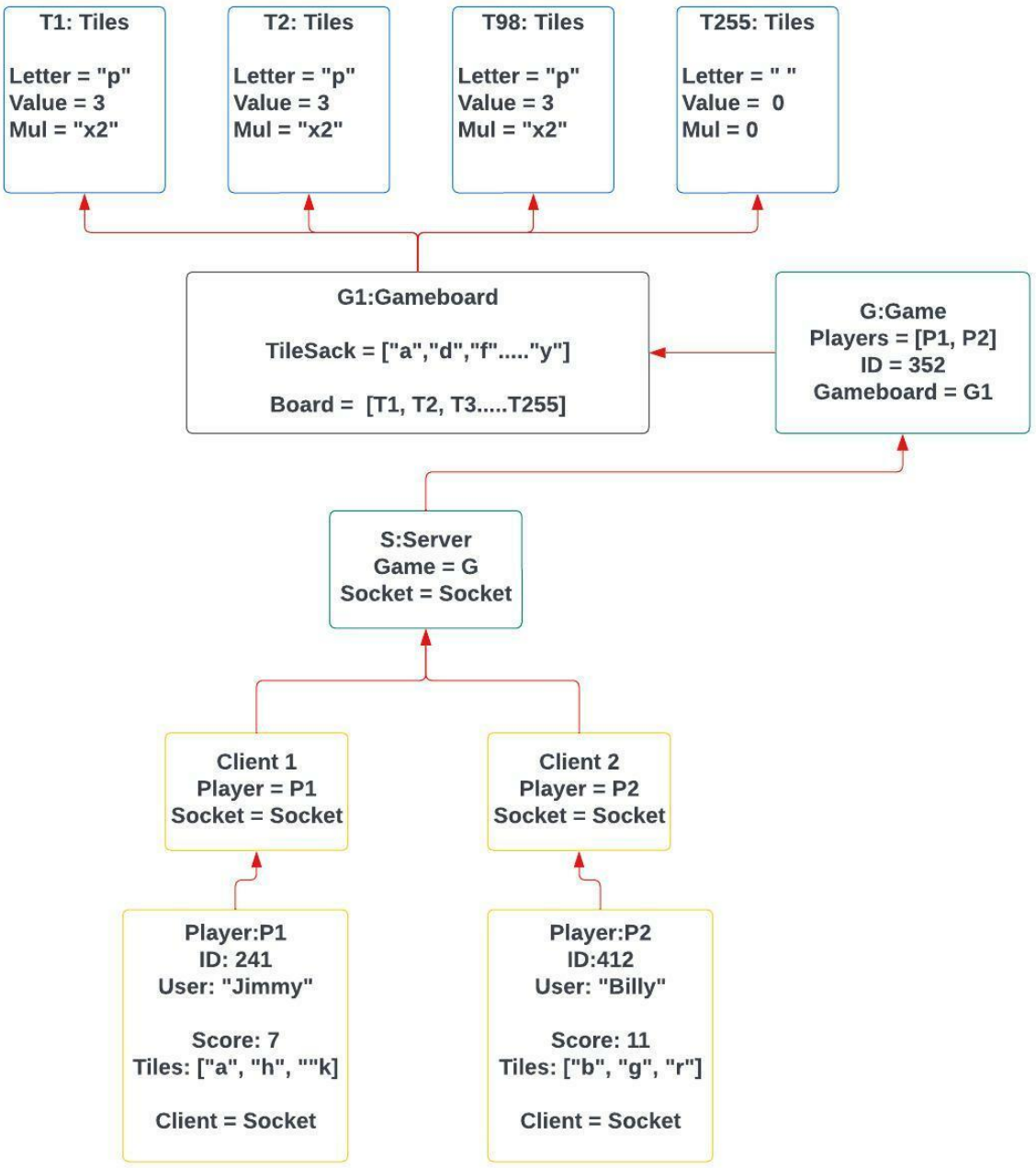
Make Move



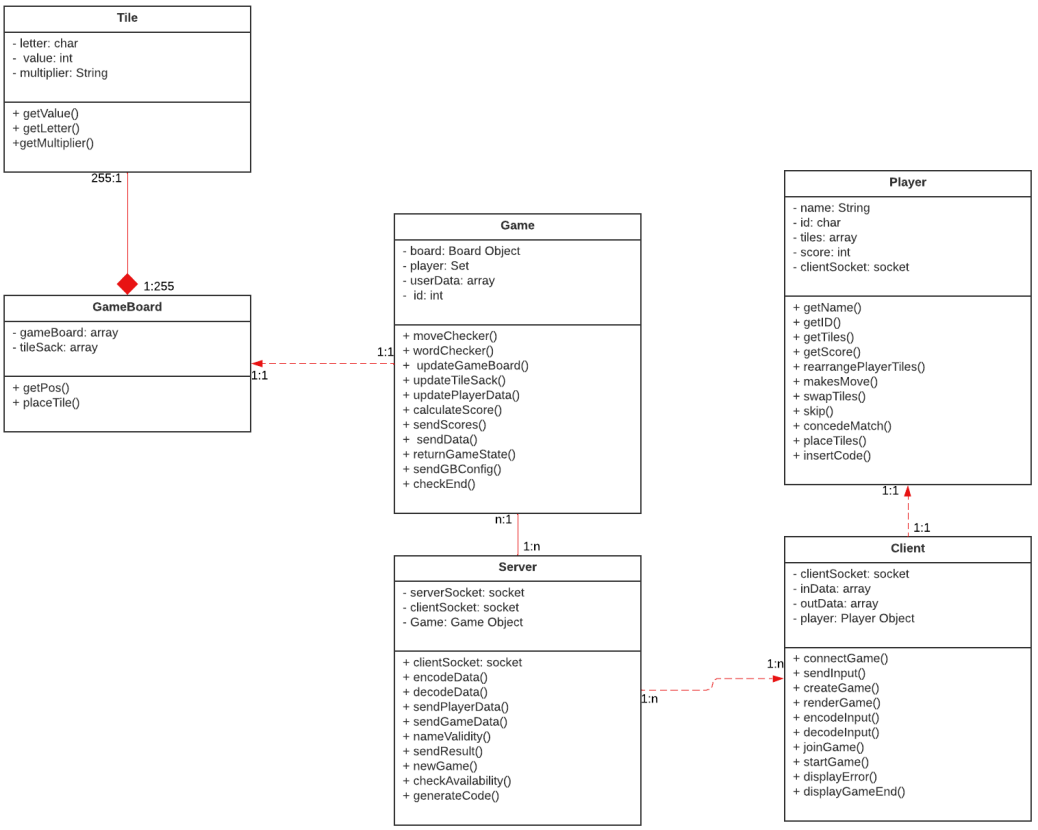
Game End



Revised Object Diagrams



More Refined Class Diagrams



Class Skeletons

```
class Server:
    def __init__(self, socket, client_socket, game):

    self.socket = socket      #Socket parameter: object for the server.
    self.clients = clients    #Clients parameter: List of client objects
    self.game = game          #Game parameter: object that handles the
logic.

    def encode(self, data):   #Data parameter: data transferred via
socket to the Game or Client object.
        return encoded_data

    def decode(self, data):   #Re-formats data to be parsed and used.
        return decoded
```

```
    def sendPlayerData(self, data, target): #Sends player data to Game
or Client.
        return #void

    def sendGameData(self, data, target):       # Sends game data to
Game or Client.
        return

    def nameValidity(self, name):       # Checks if name conforms to
limits and if one exists.
        return

    def sendResult(self):       # Sends final results.
        return

    def newGame(self):          # Creates a new game instance.
        return #void

    def checkAvailability(self): #Checks if another game instance can
be created.
        return #bool

    def generateCode(self):     # Generates invite code for game
instance.
        return

class Client:
    def __init__(self, socket, in_data, out_data, player):

    self.socket = socket      #Socket parameter: object for the client.
    self.in_data = in_data    #In-data parameter: JSON object.
    self.out_data = out_data  #Out-data parameter: JSON object.
    self.player = player      #Player parameter: player object.

    def connectGame(self, code, client): #Connects client to the game
        return #void

    def sendInput(self, data, client):   #Sends data to client.
        return #void

    def createGame(self, name):
        return #void
```

```
    def renderGame(self,name): #Creates a game.
        return #void

    def encode(self, input):   #Data parameter: data transferred via
socket to the Game or Client object.
        return encoded_input

    def decode(self, input):   #Re-formats data to be parsed and used.
        return decoded_input

    def joinGame(self, code, target):              # Joins the game
utilising the game code.
        return #void

    def startGame(self):           # Sends start game request to
server.
        return  #void

    def displayError():        # Displays error that occurred.
        return

    def displayGameEnd():      # Displays game ending screen.
        return


class Player:
    def __init__(self, ID, tiles, score, client_socket):

    self.ID = ID                        #Id parameter: player unique id.
    self.tiles = tiles                  #Tiles parameter: empty list.
    self.score = score                  #Score parameter: player score =
0.
    self.client_socket = client_socket #Client_socket parameter:
player's client socket.

    def getName(self):     #Returns players Name.
        return self.name

    def getID(self):       #Returns ID.
        return self.ID

    def getTiles(self):    #Returns player's tiles.
        return self.tiles
```

```
    def getScore(self):    #Returns player's score.
        return self.score

    def rearrangePlayerTiles(self, tiles):      #Allows player to
rearrange their tiles on hand.
        return

    def makesMove(self):        # Player makes a move.
        return #void

    def swapTiles(self):   #Swaps player's tiles.
        return #void

    def skip(self):        #Skips player's turn.
        return #void

    def concedeMatch(self):#Player concedes match.
        return #void

    def placeTiles(self):  #Player places tile.
        return #void

    def insertCode():
        return


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

    self.board = board         #Board parameter: board object.
    self.player = player       #Player parameter: player objects.
    self.user_data = user_data #User_data parameter: user data.
    self.ID = ID               #ID parameter: game id.

    def move_checker(self):       #Checks for illegal moves.
        return #bool

    def word_checker(self, word): #Checks dictionary if word is legal.
        return #void

    def update_game_board(self):  #Updates board configuration
        return #void
```

```
    def update_tile_sack(self):   #Updates the tile sack.
        return #void

    def update_playerdata(self):      #Updates user data.
        return #void

    def sendScores(self):               #Sends final scores
        return self.players

    def calculate_score(self):    #Calculates word score.
        return score

    def send_data(self):          #Sends updated game object to server.
        return #void

    def returnGameState(self):      # Sends an updated game state.
        return state

    def sendGBConfig(self, board):
        return #bool

    def checkEnd(self):             #Checks if end conditions have been
met.
        return #bool

class Tile:
    def __init__(self, letter, value, multiplier):

    self.letter = letter        #Letter parameter: string/char of a
letter (A-Z).
    self.value = value          #Value parameter: value of letter.
    self.multiplier = out_data  #Multiplier parameter: multiplier type
of the tile.

    def get_value(self):        #Returns value attribute to tile
object.
        return self.value

    def get_letter(self):       #Returns letter attribute to tile
object.
        return self.letter
```

```
    def get_multiplier(self):    #Returns multiplier attribute to tile
object.
        return self.multiplier


class GameBoard:
    def __init__(self, array, tile_sack):

    self.array = array           #Array parameter: python list
representing the board.
    self.tile_sack = tile_sack   #Tile_sack parameter: array with random
tile objects.

    def get_pos(self, pos):      #Returns tile object at a position on
the board.
        return array[pos]

    def place_tile(self):        #Places tile onto the board.
        return array
```

Minutes/Notes of team meetings

4th MEETING

Minutes for 11/10/2022, Tuesday 12:30 pm

Members Present:

- Kline
- Mustafa
- Matt
- Ronghui
- Vaidas
- Effa
- Liam

Members Absent:

- N/A

What did we do/discuss?:

During the meeting, we agreed to complete sections 1-5 within the first two weeks. We assign one person per section. We aim to have these done in 2 weeks before progressing onto the remaining sections.

What are we going to do?

For now, we want to look at the rubric and see how our group can achieve the maximum amount of marks available. Each person will aim to at least be halfway finished with their section before the next meeting.

Any difficulties?

- None

5th MEETING

Minutes for 18/10/2022, Tuesday 12:00 pm

Members Present:

- Kline
- Mustafa
- Matt
- Ronghui
- Vaidas
- Effa
- Liam

Members Absent:

- N/A

What did we do/discuss?:
Discussed the completion progress of each task.
Discussed any ambiguities regarding the first 6 tasks.
Created a checklist for tasks + additional helpful resources.
We discussed the feedback given by Renaat for the last submission.

What are we going to do?
Try to finish off the first 6 tasks by the next meeting.
Ask for help if needed in Discord.

Any difficulties?

- None

6th MEETING

Minutes for 25/10/2022, Tuesday 12:00 pm

Members Present:

- Kline
- Mustafa
- Matt
- Ronghui
- Vaidas

Members Absent:

- Effa
- Liam

What did we do/discuss?:
Discussed the completion progress of each task.
Updating the task checklist.
Discussed any ambiguities regarding the last 5 tasks and assigning people to each one.

What are we going to do?
Try to finish off the last 5 tasks by the next meeting.
Ask for help if needed in Discord.

Any difficulties?

- None