MY SCRABBLE

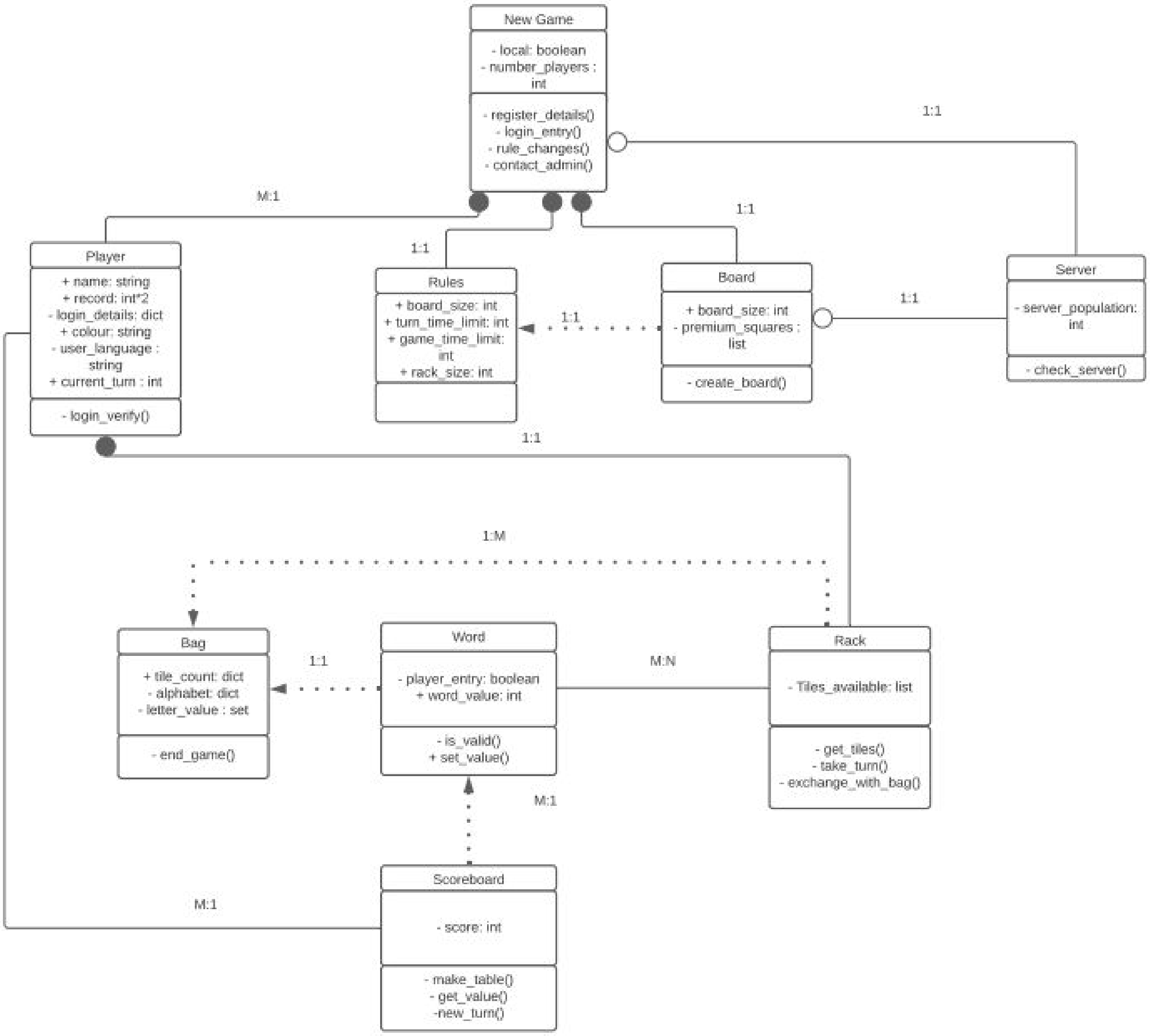
CA314

Product/Class Design

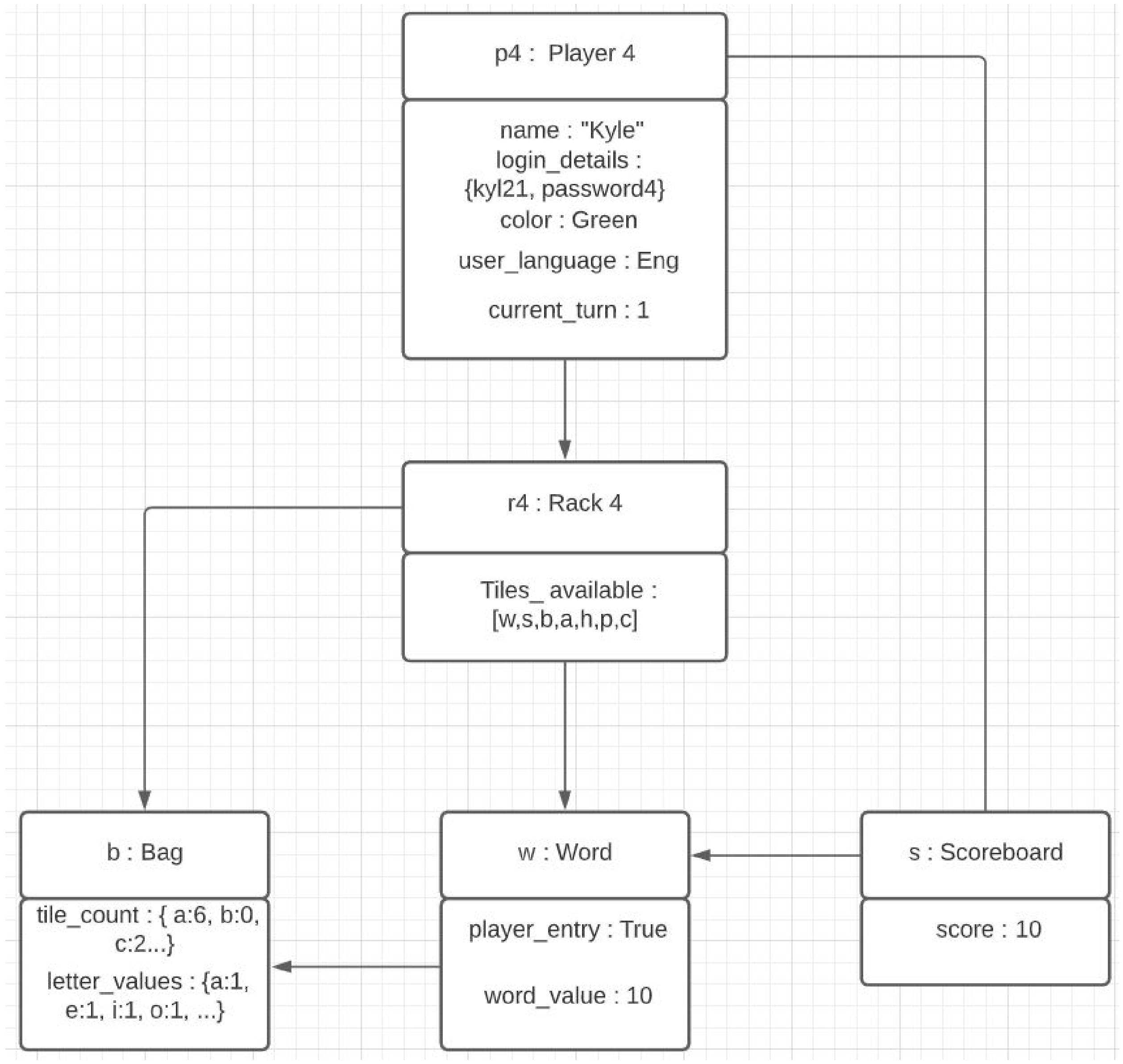
Student Name	Student Number
Kevin Cogan	18421694
Alex O’Neil	18414882
Cameron Fitzpatrick	18493006
Niall Dagg	17311161
Conor Mckeon	17386616
Jeff Thomas	18105319

Table Of Contents

1. Refined Class Diagram

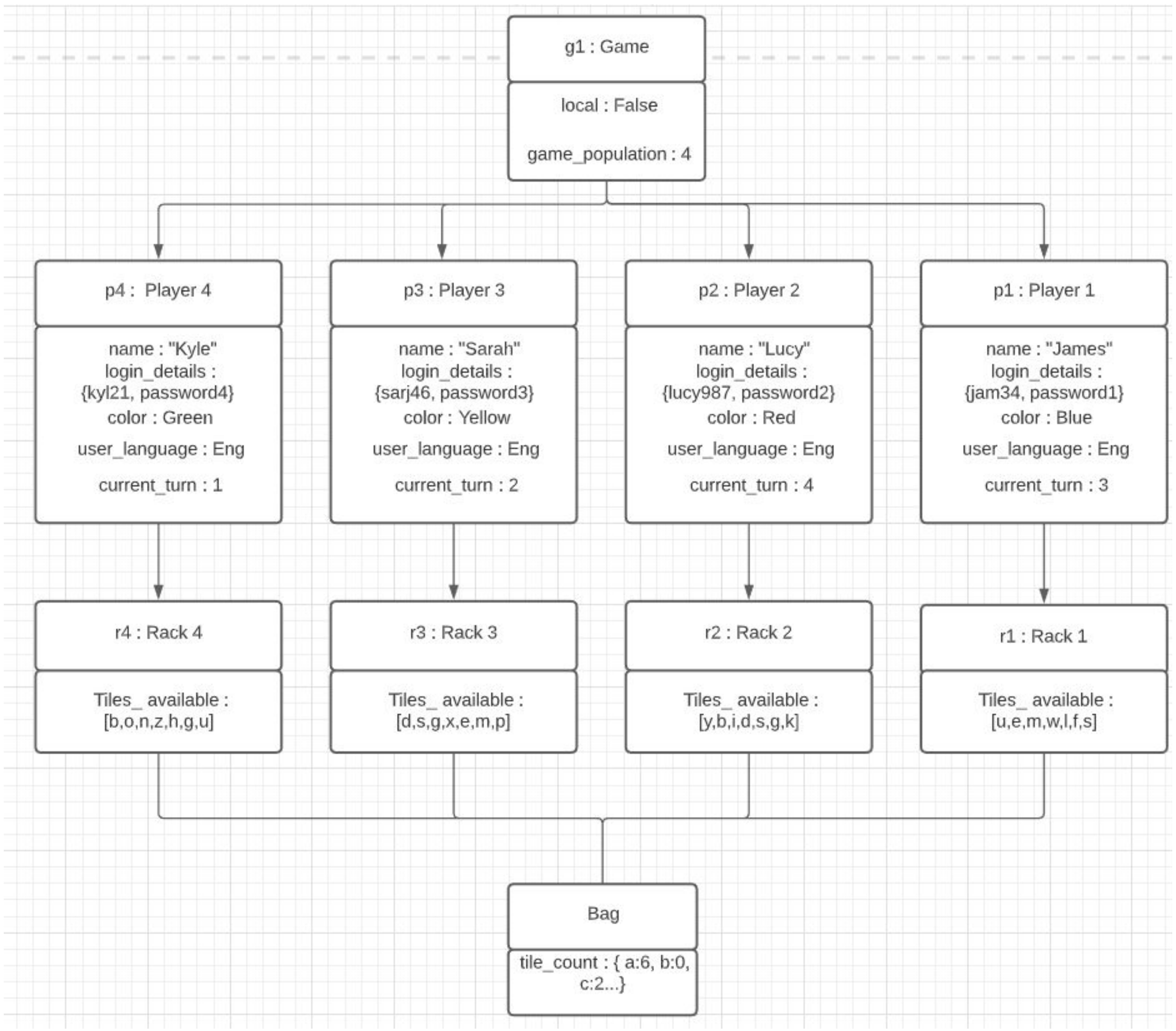


2. Object Diagrams



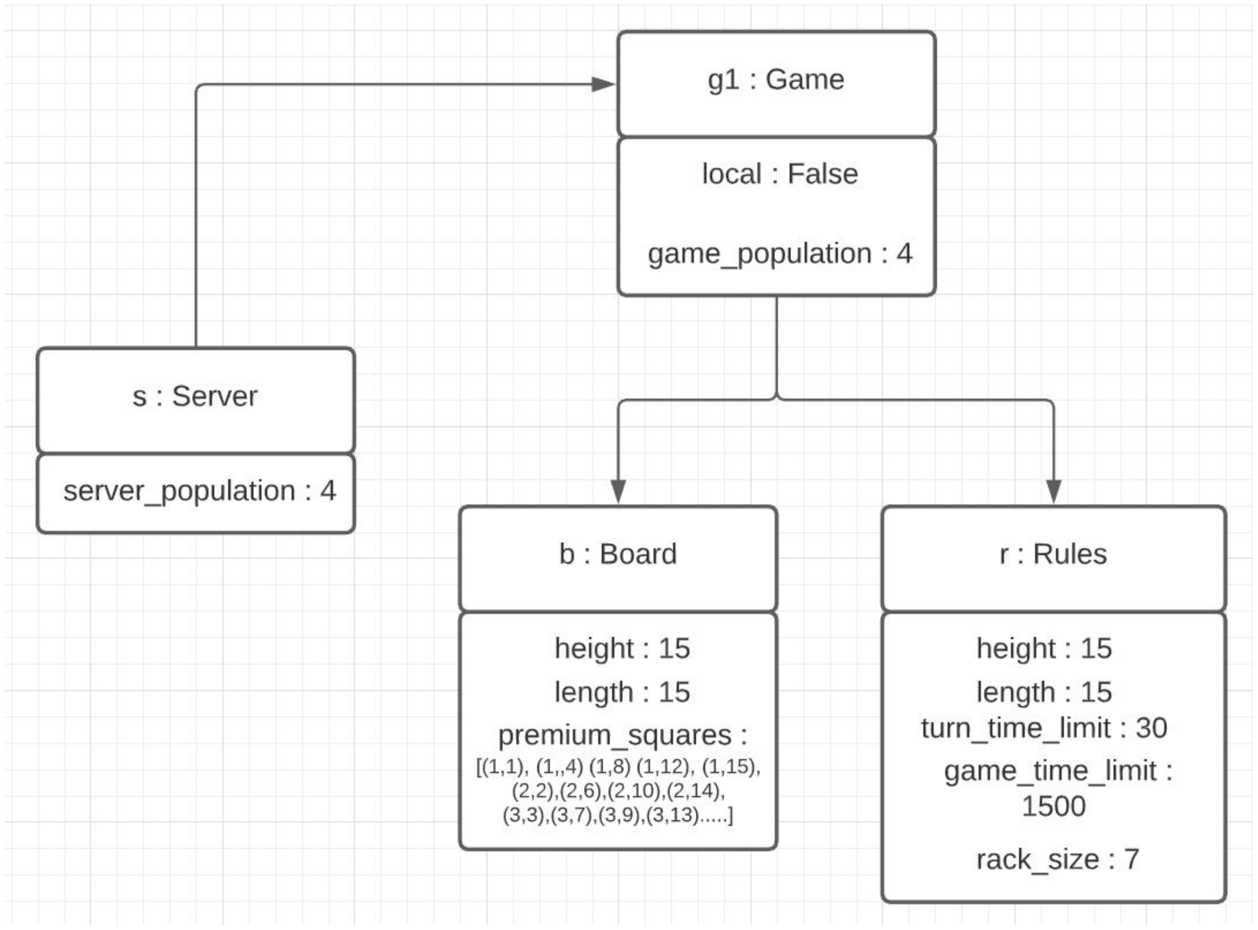
2.1 Online Game Initialization

The classes that begin our game of Scrabble are obviously very important, we cannot play a game without them. Our server-class communicates to the game class, game population and online/offline status are established. Our Rules class allows players to choose their desired game rules such as turn and game time limits, rack sizes and board size.



2.2 Setting up players and racks

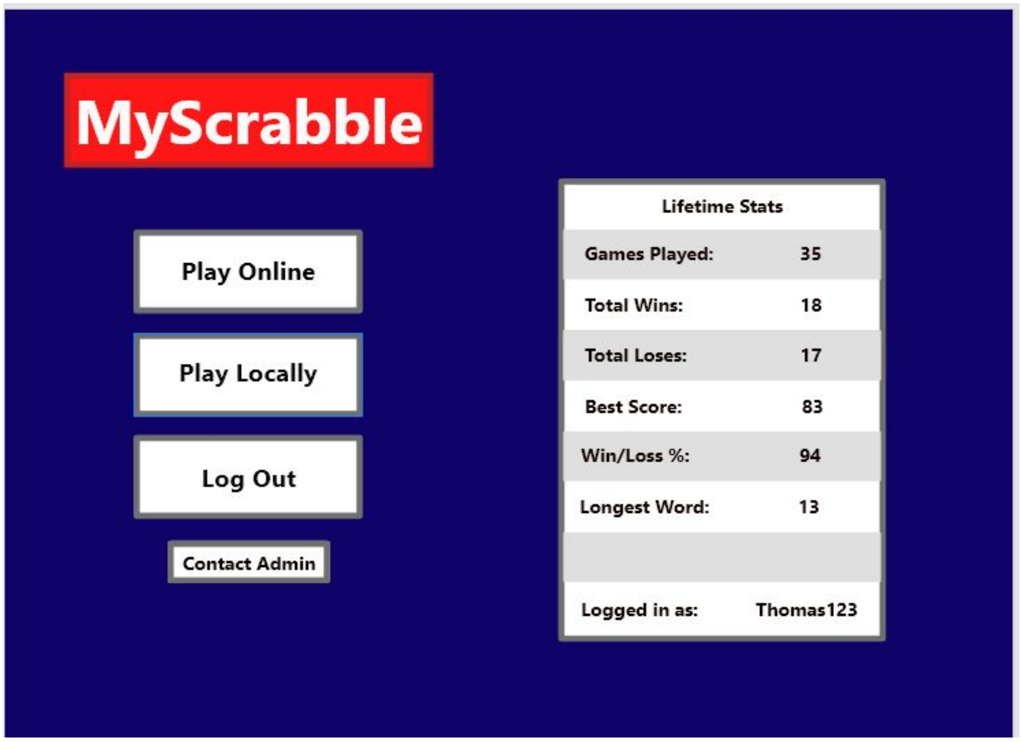
Once the game is set up, our players are established and are assigned attributes such as name, colour and their turn order in the game. Their chosen language and login attributes are included but are not visible. Players are all assigned tile racks that have their tiles_available attributes populated by the bag class. Our bag class is keeping track of total tile_count remaining, along with the values of each of those tiles should they be played.



<u>2.3 The player plays a word</u>

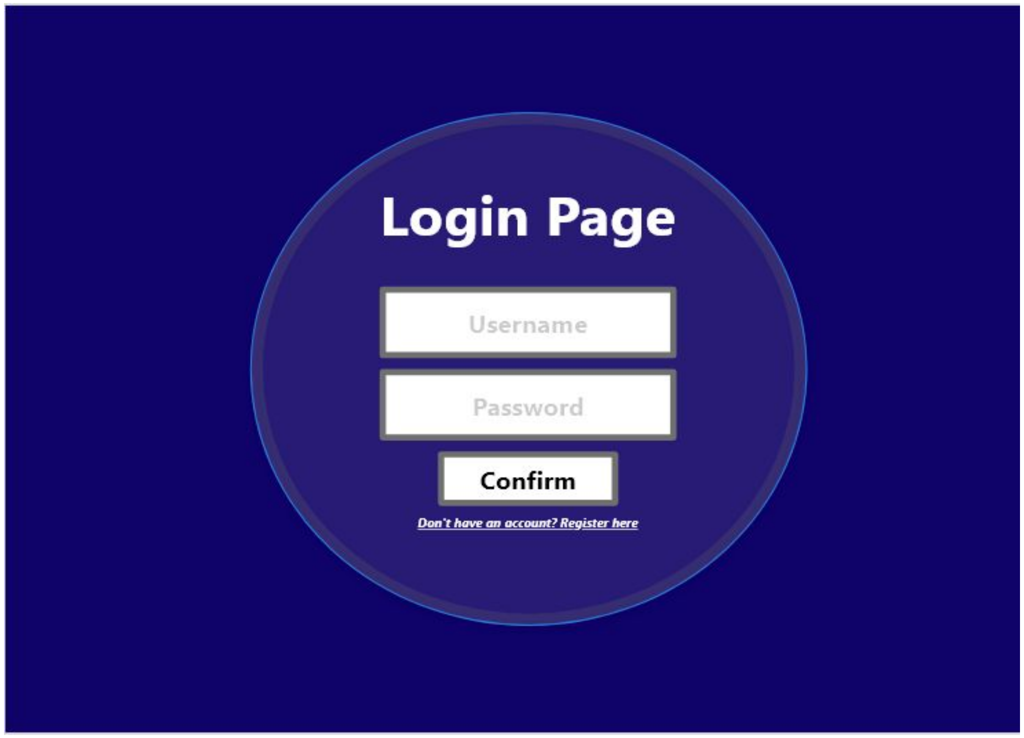
Obviously, if our players cannot play words, no game can be played, so this section of the classes is also very important. Our player submits a word from their Rack to the Word class, where it is checked for validity and assigned a point value using the letter_values attribute from Bag. Once both attributes are secured, they are sent to Scoreboard where the score is added to the scoreboard table and displayed back to Player.

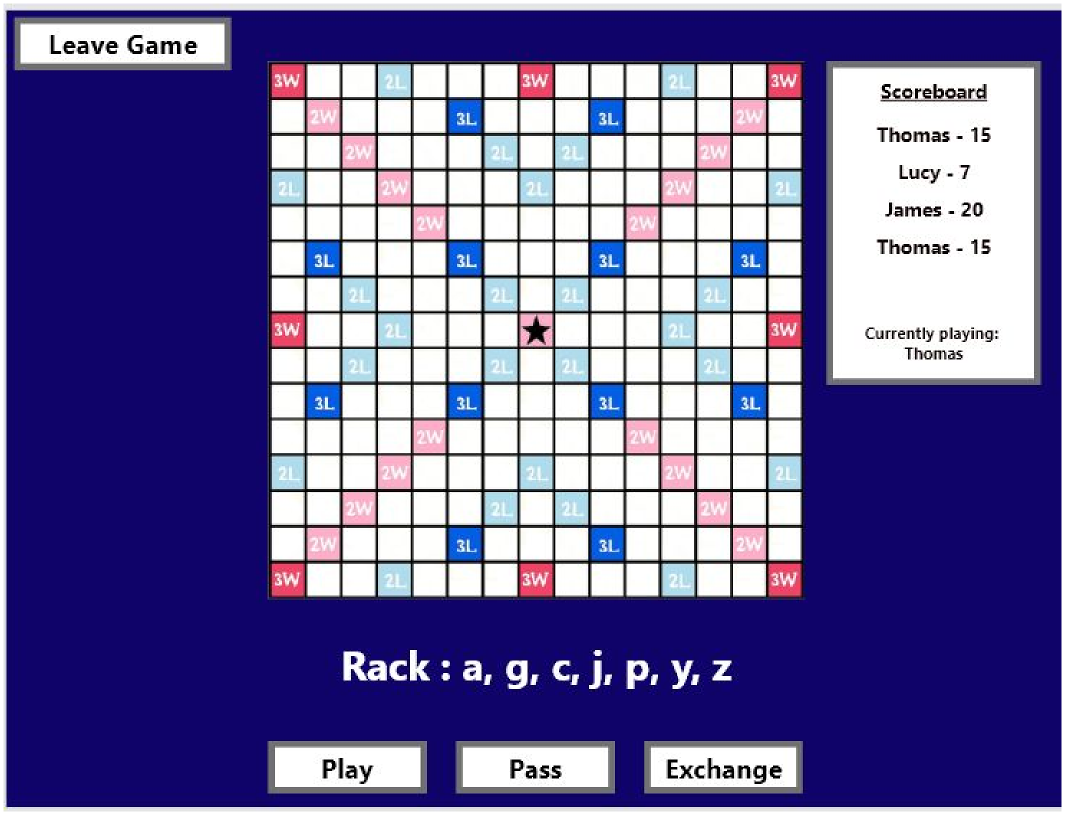
3. UI Mockups



Lifetime Stats	
Games Played:	35
Total Wins:	18
Total Loses:	17
Best Score:	83
Win/Loss %:	94
Longest Word:	13
Logged in as:	Thomas123

3.1 Homepage ↗ ↙ 3.2 Log-In Page



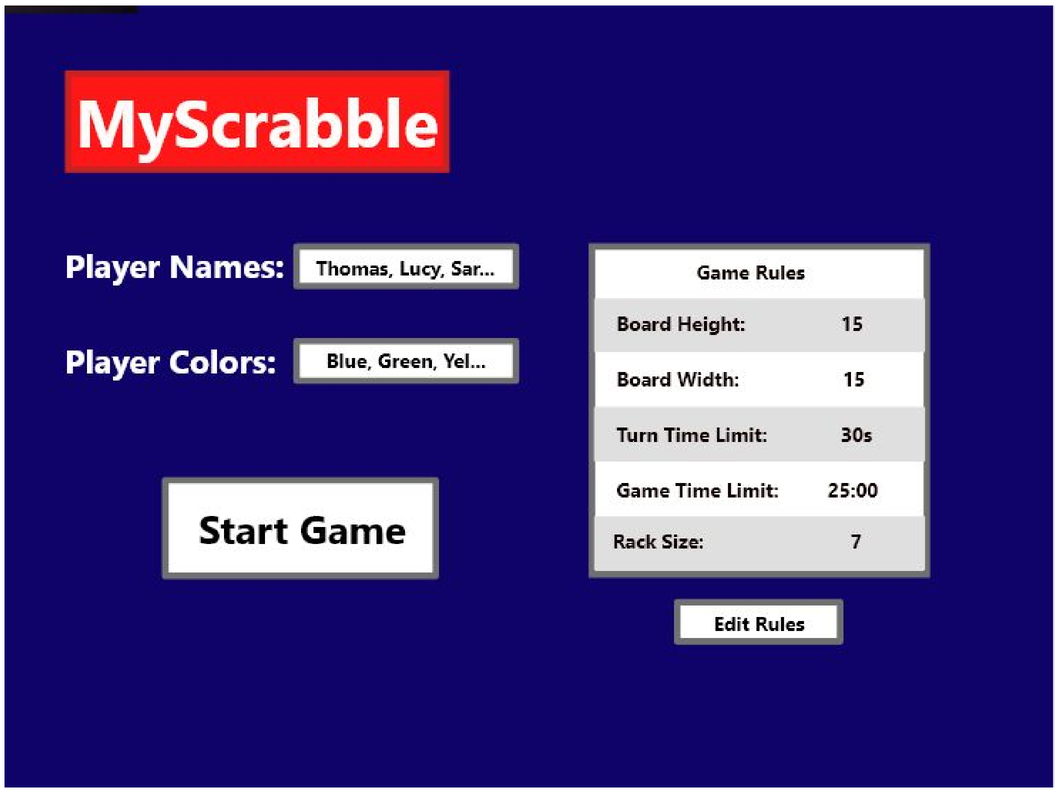


Thomas - 15
Lucy - 7
James - 20
Thomas - 15

Currently playing: Thomas

Rack : a, g, c, j, p, y, z

3.3 In-Game Screen ↗ ↙ 3.4 Offline Game Setup



Player Names: Thomas, Lucy, Sar...

Player Colors: Blue, Green, Yel...

Game Rules	
Board Height:	15
Board Width:	15
Turn Time Limit:	30s
Game Time Limit:	25:00
Rack Size:	7

4. Client-Server Experiments

The Initial Network testing for this application was done using a basic tkinter window that was used to make http requests to a restful flask api. The tkinter client component used the requests library in python to generate multiple http requests to the endpoints of the flask application.

Flask was chosen for this purpose due to the group's experience in python as well as the availability of multiple modules within flask that allow for easy integration for session management, cors, sanitizing inputs etc.

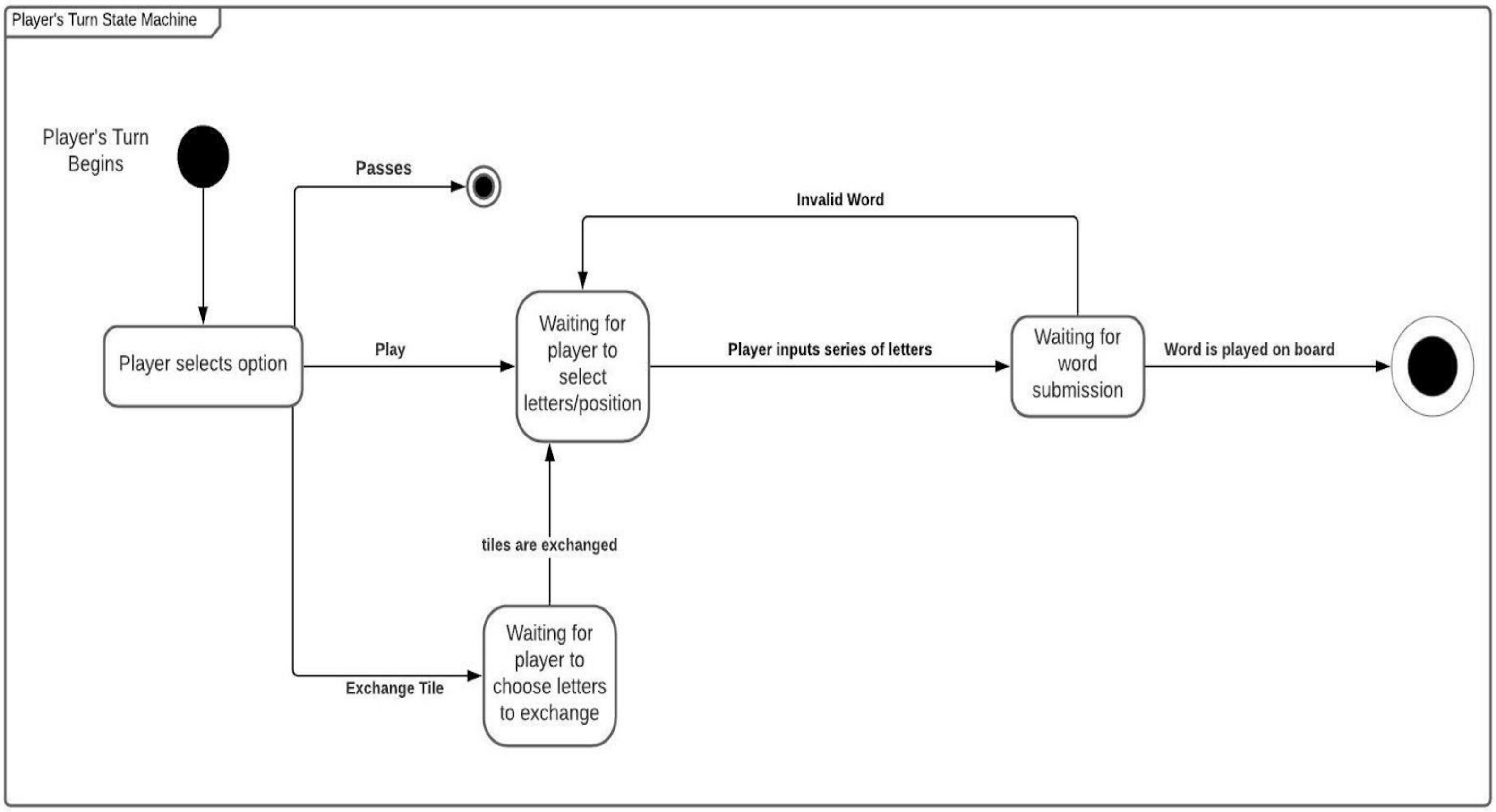
The flask application itself was running on an Apache Server with a WSGI interface. For our authentication database, we used a postgresql instance and managed to integrate it with our flask application. Within testing parameters, we managed to send and receive information in the client from the api via json objects. We were also able to successfully authenticate via the authentication endpoints and creating user sessions. The experiment was concluded with the testing of two client instances whereby both instances shared the same copy of a proposed board and moves made one client were able to register in the other via the networking setup. Altogether this configuration seemed to be quite effective in communicating between our python based client.

For alternatives, a browser based component was considered but since the timeline of the project did not allow for the integration of certain javascript front end frameworks, tkinter was chosen as the visual interface substitute. However, the flask api itself can be configured with a browser based frontend due to the api's independent nature from the client, allowing for further expansion if need be.

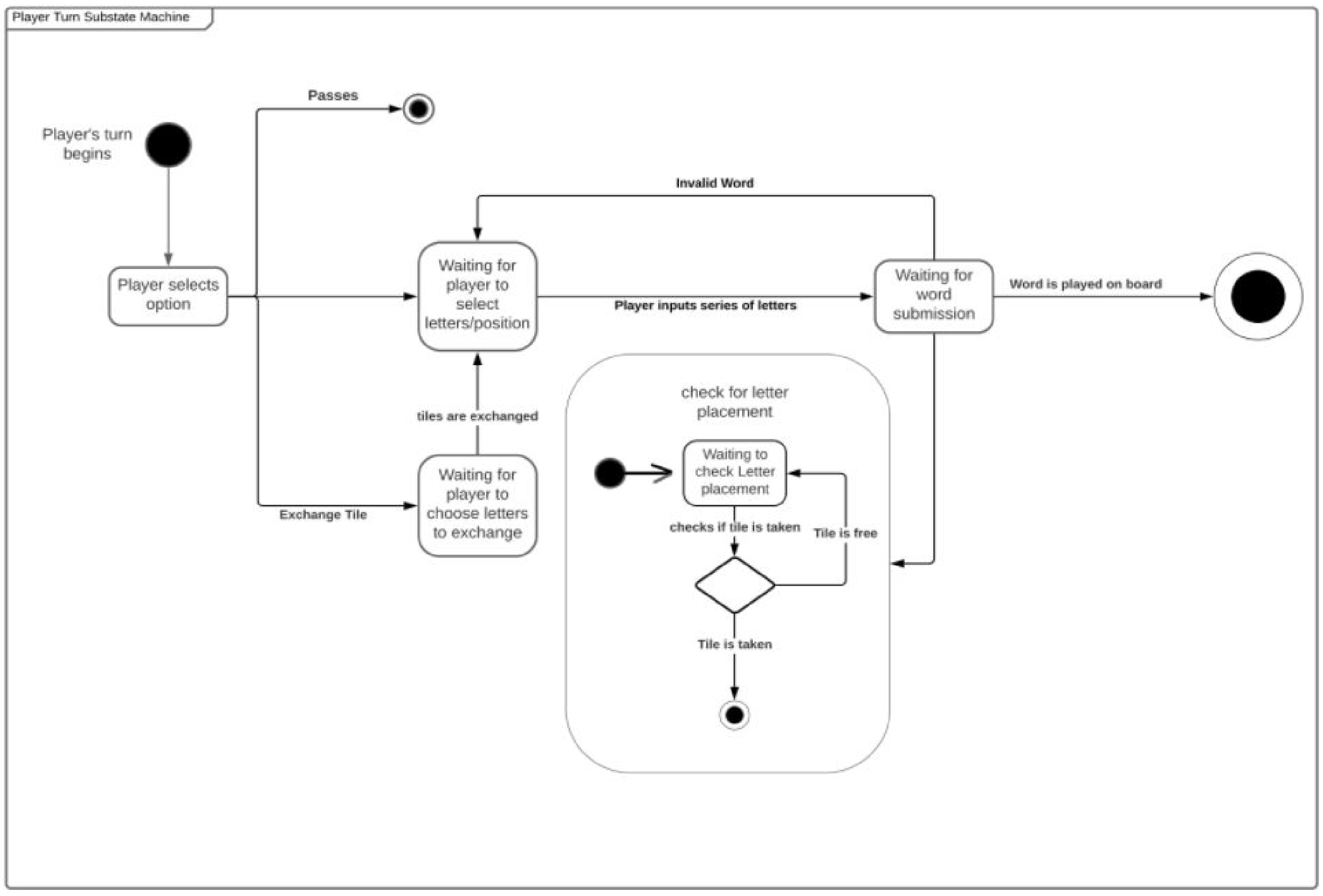
Therefore for the network configuration of our application, we will be creating our client side software in python using tkinter for the graphical interface. The client software will be in direct conversation with the flask api either using a socket or plain http requests. The api itself which will represent our game server will keep track of user sessions, game state and rooms. Postgresql will be used as our authentication database for multiplayer games. The game server will be hosted on an Apache instance that will have WSGI to support our python based application.

4. State machines

4.1 Player's Turn State Machine

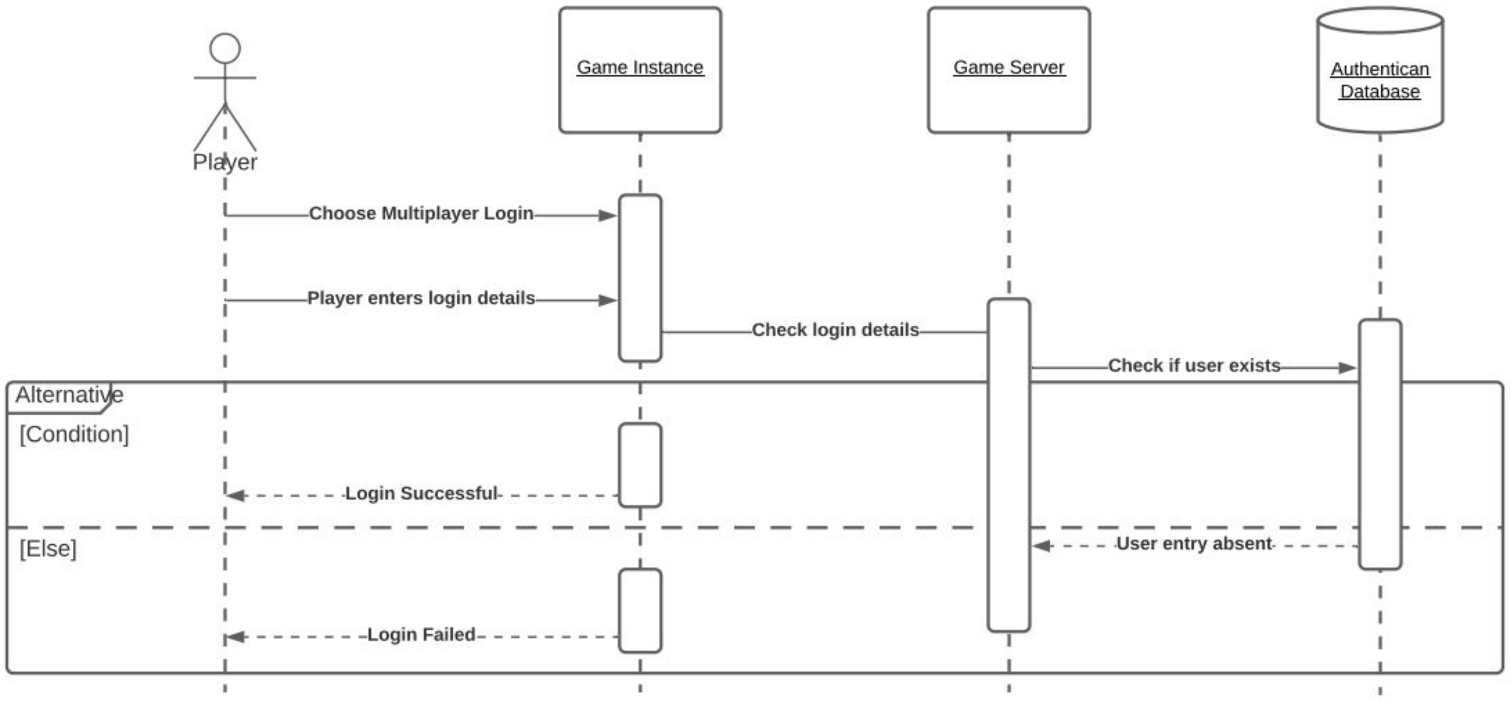

Player's Turn State Machine
Player's Turn Begins
Passes
Invalid Word
Player selects option
Play
Waiting for player to select letters/position
Player inputs series of letters
Waiting for word submission
Word is played on board
tiles are exchanged
Exchange Tile
Waiting for player to choose letters to exchange

4.2 Player's Turn Substate Machine

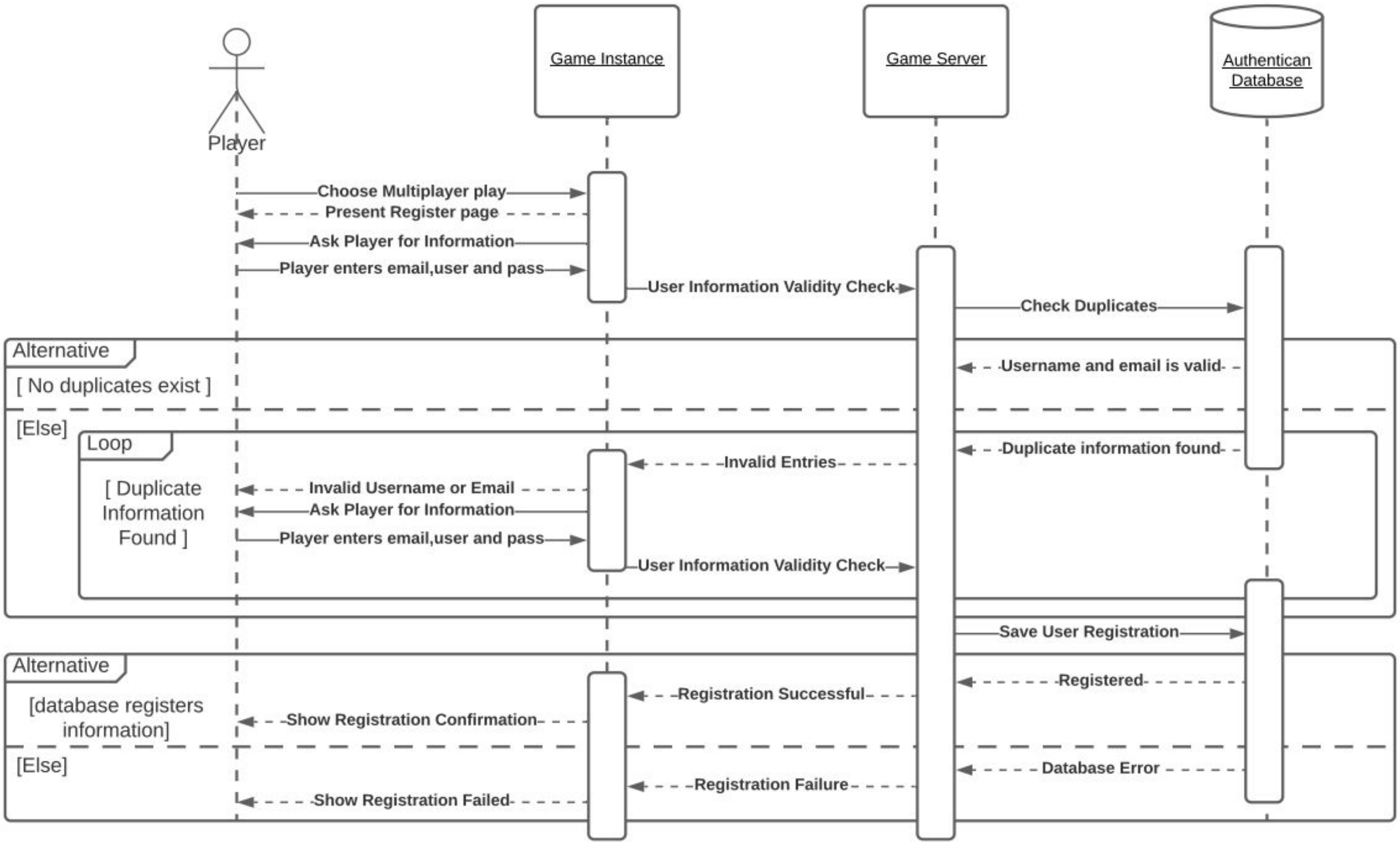

Player Turn Substate Machine
Player's turn begins
Passes
Invalid Word
Player selects option
Waiting for player to select letters/position
Player inputs series of letters
Waiting for word submission
Word is played on board
tiles are exchanged
Exchange Tile
Waiting for player to choose letters to exchange
check for letter placement
Waiting to check Letter placement
checks if tile is taken
Tile is free
Tile is taken

5. Sequence Diagrams

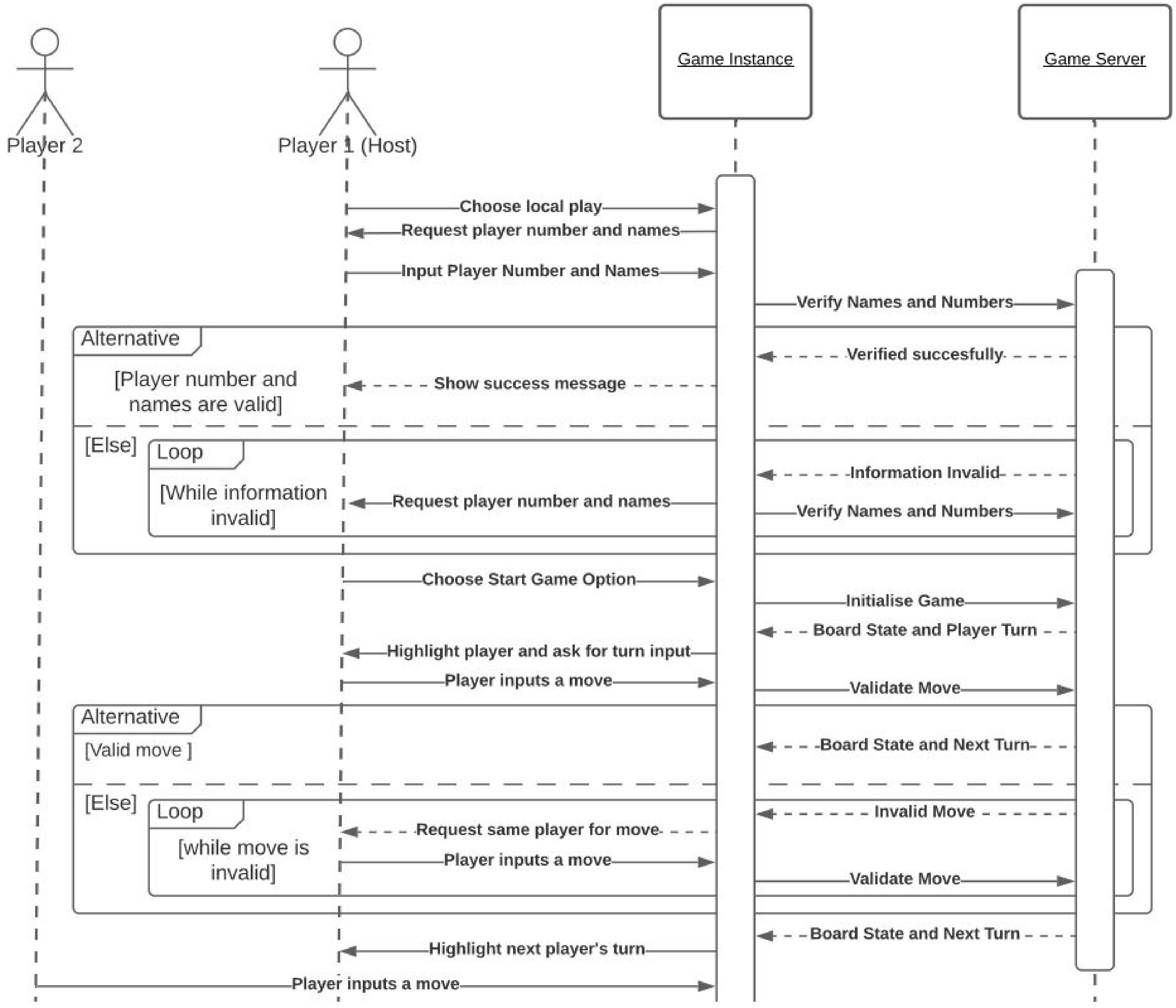
5.1 Login Diagram



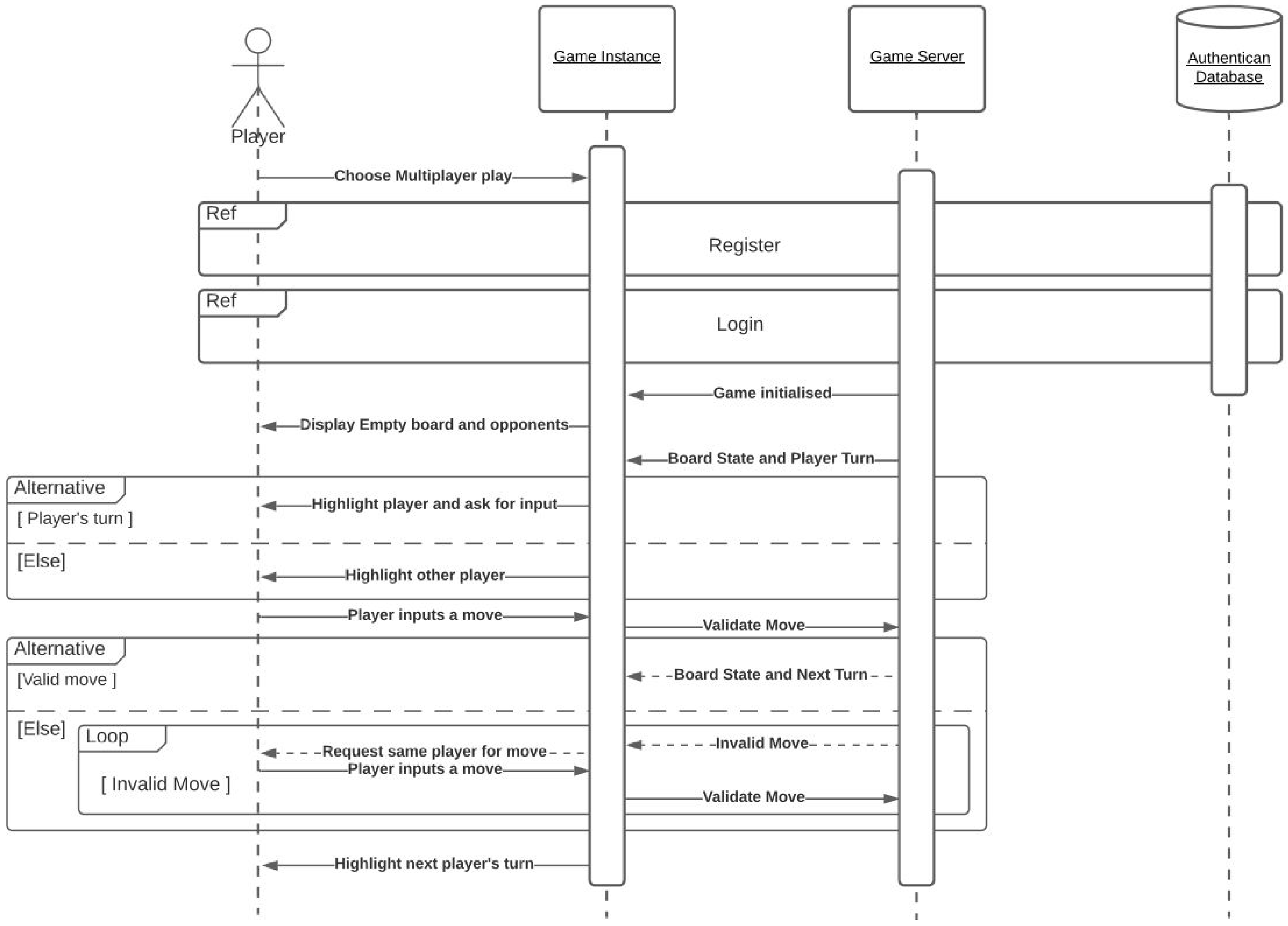
5.2 Register Diagram


Game Instance
Game Server
Authentican Database
Player
Choose Multiplayer play
Present Register page
Ask Player for Information
Player enters email,user and pass
User Information Validity Check
Check Duplicates
Alternative
[No duplicates exist]
Username and email is valid
[Else]
Loop
Duplicate information found
Invalid Entries
[Duplicate Information Found]
Invalid Username or Email
Ask Player for Information
Player enters email,user and pass
User Information Validity Check
Save User Registration
Alternative
[database registers information]
Registered
Registration Successful
Show Registration Confirmation
[Else]
Database Error
Registration Failure
Show Registration Failed

5.3 Local Diagram

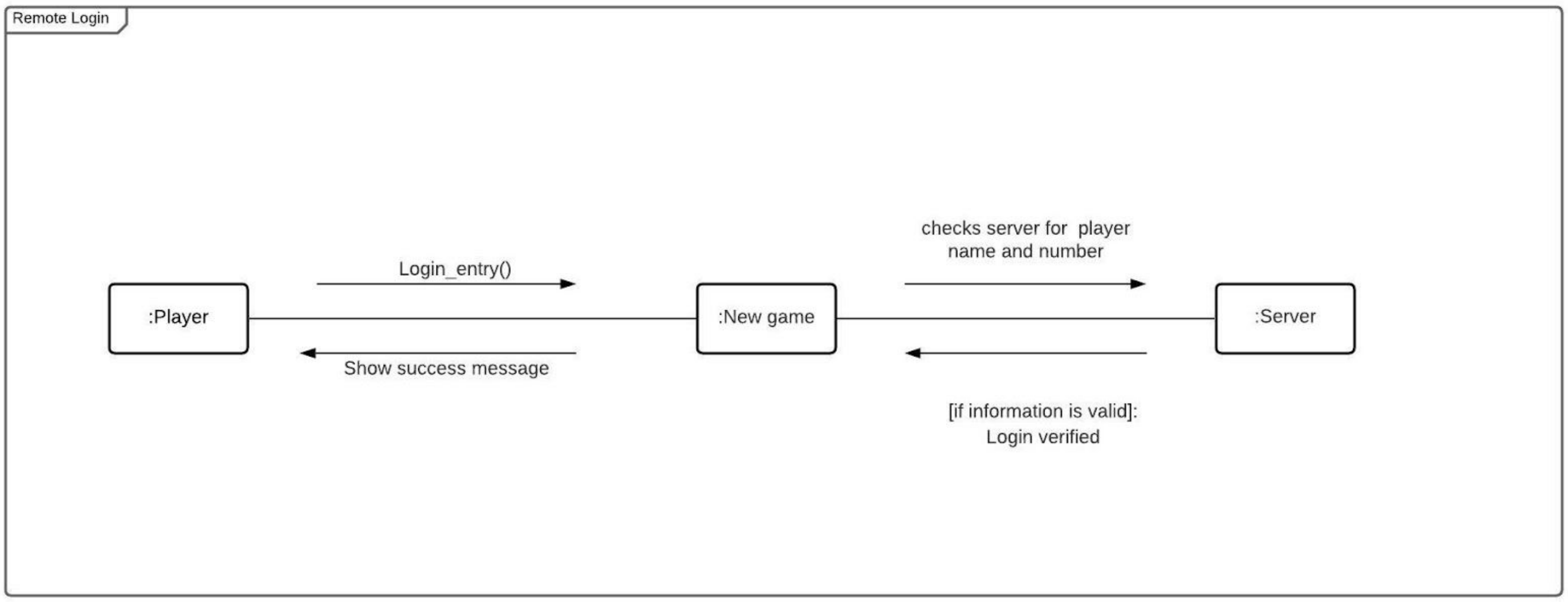


5.4 Multiplayer Diagram

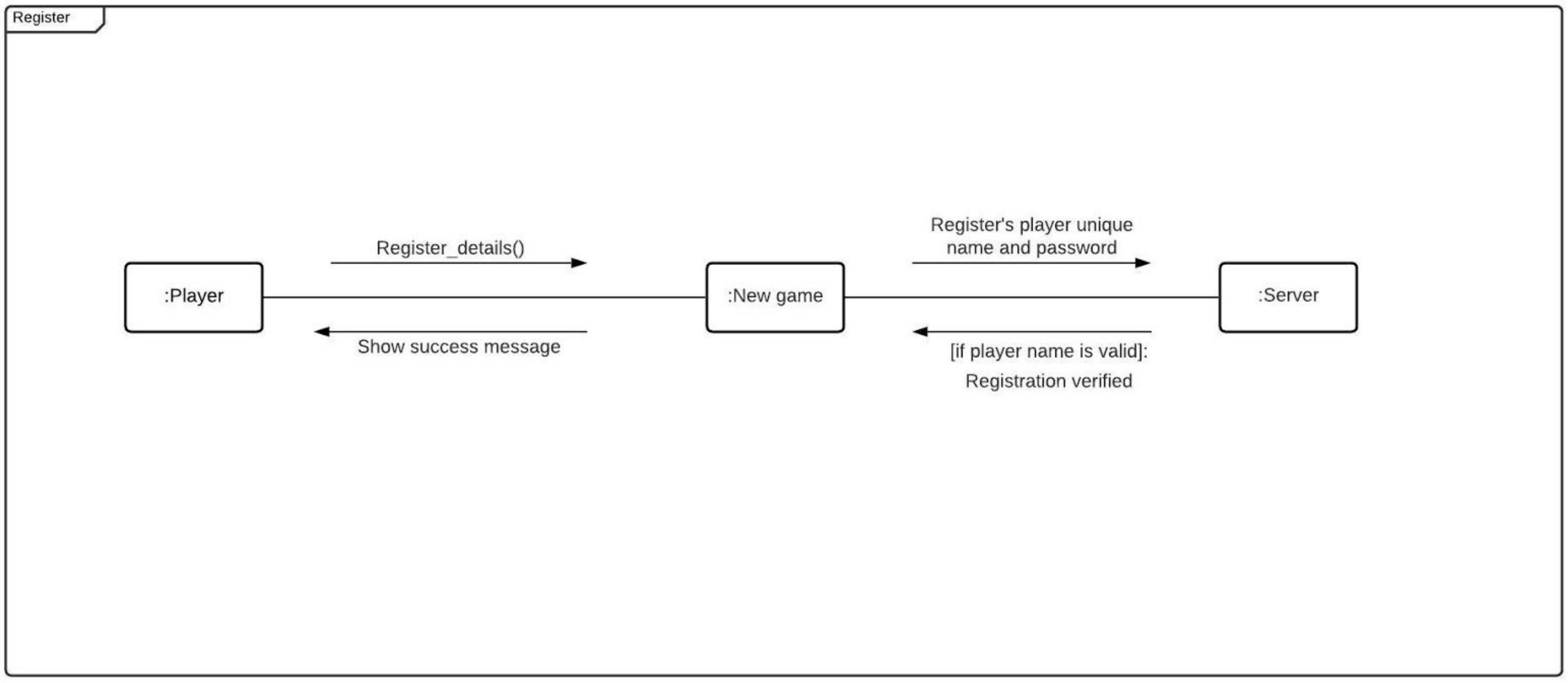


6. Collaboration Diagrams

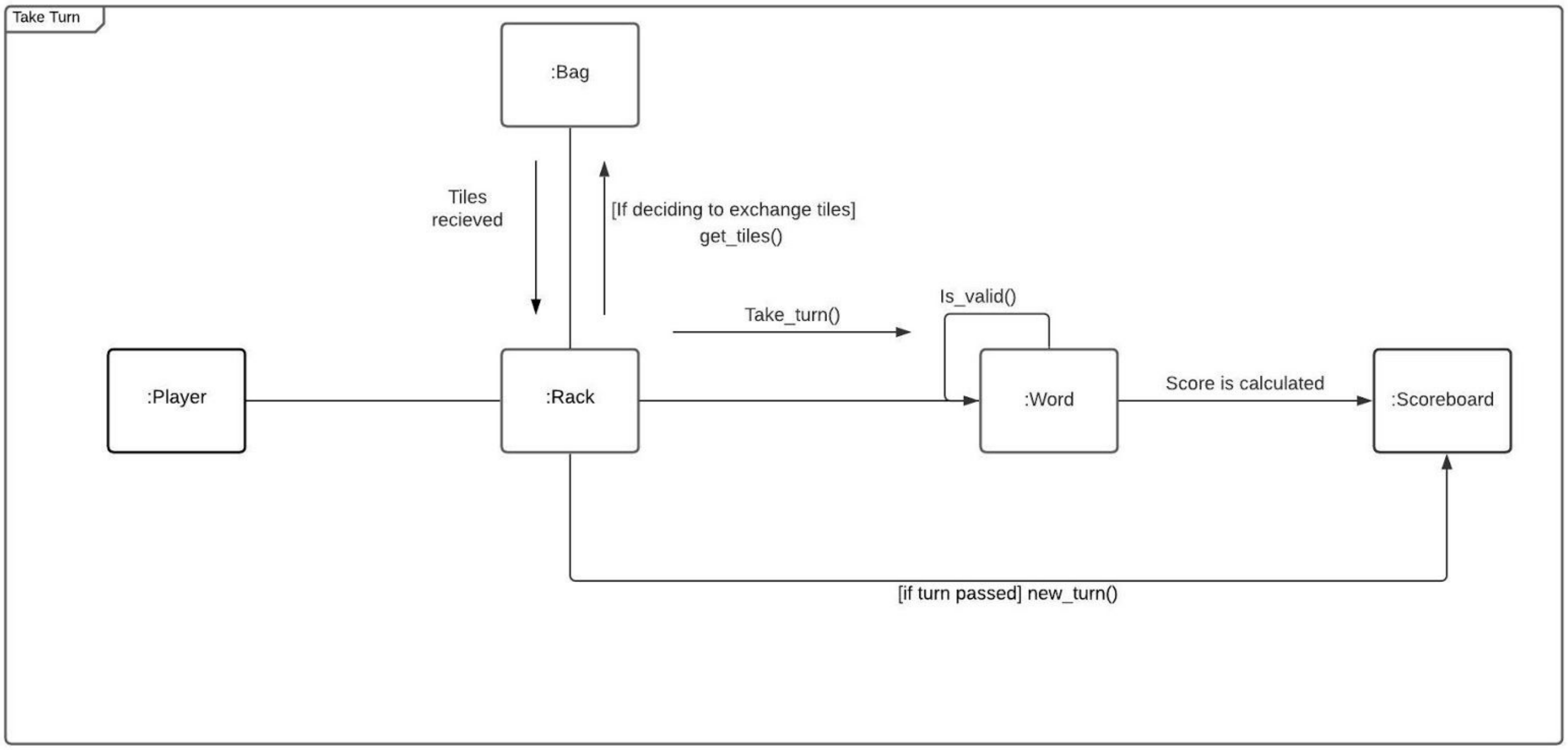
6.1 Login Diagram


Remote Login
:Player
Login_entry()
Show success message
:New game
checks server for player name and number
[if information is valid]: Login verified
:Server

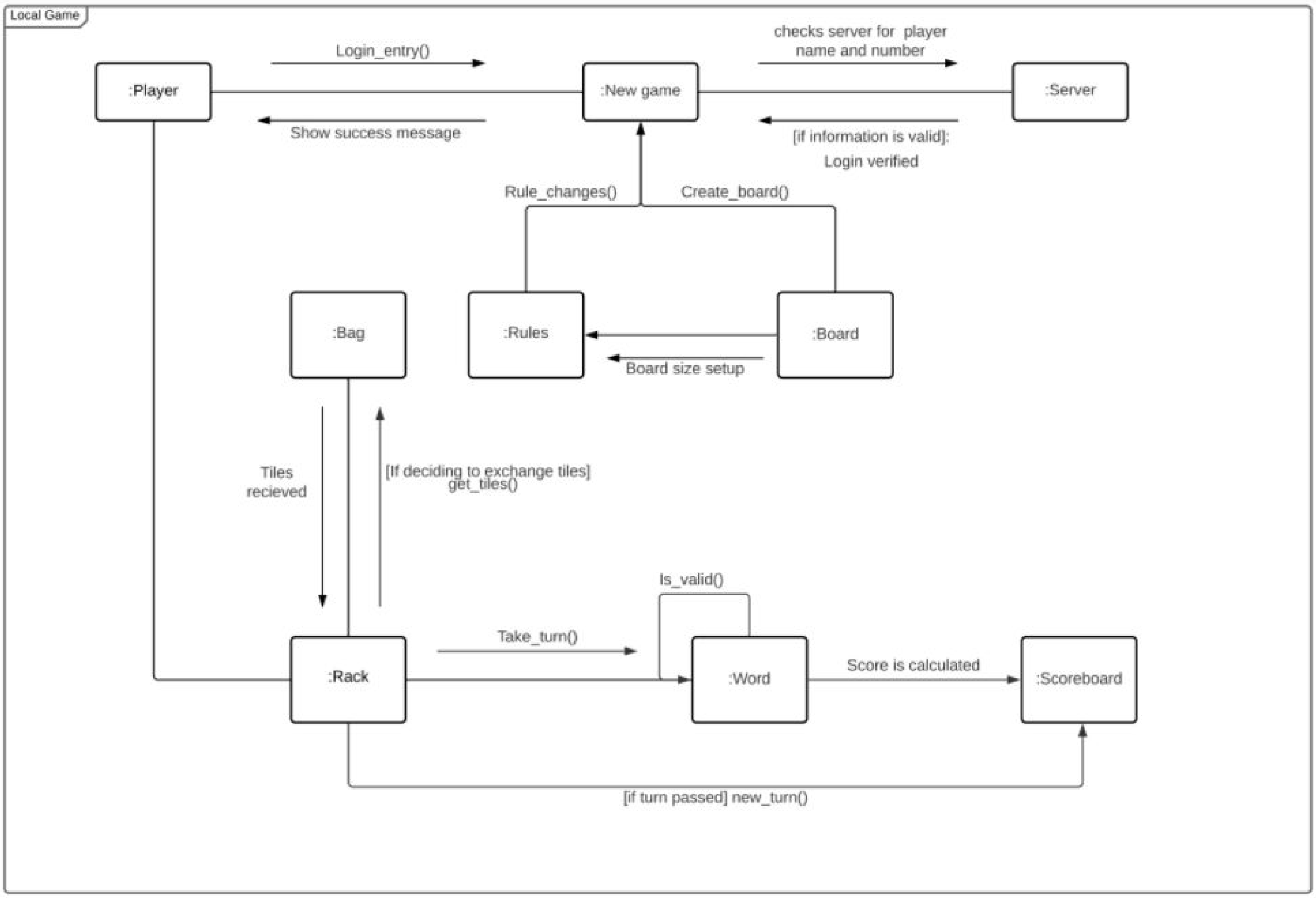
6.2 Register Diagram


Register
:Player
Register_details()
Show success message
:New game
Register's player unique name and password
[if player name is valid]: Registration verified
:Server

6.3 Take Turn Diagram for Local/Multiplayer

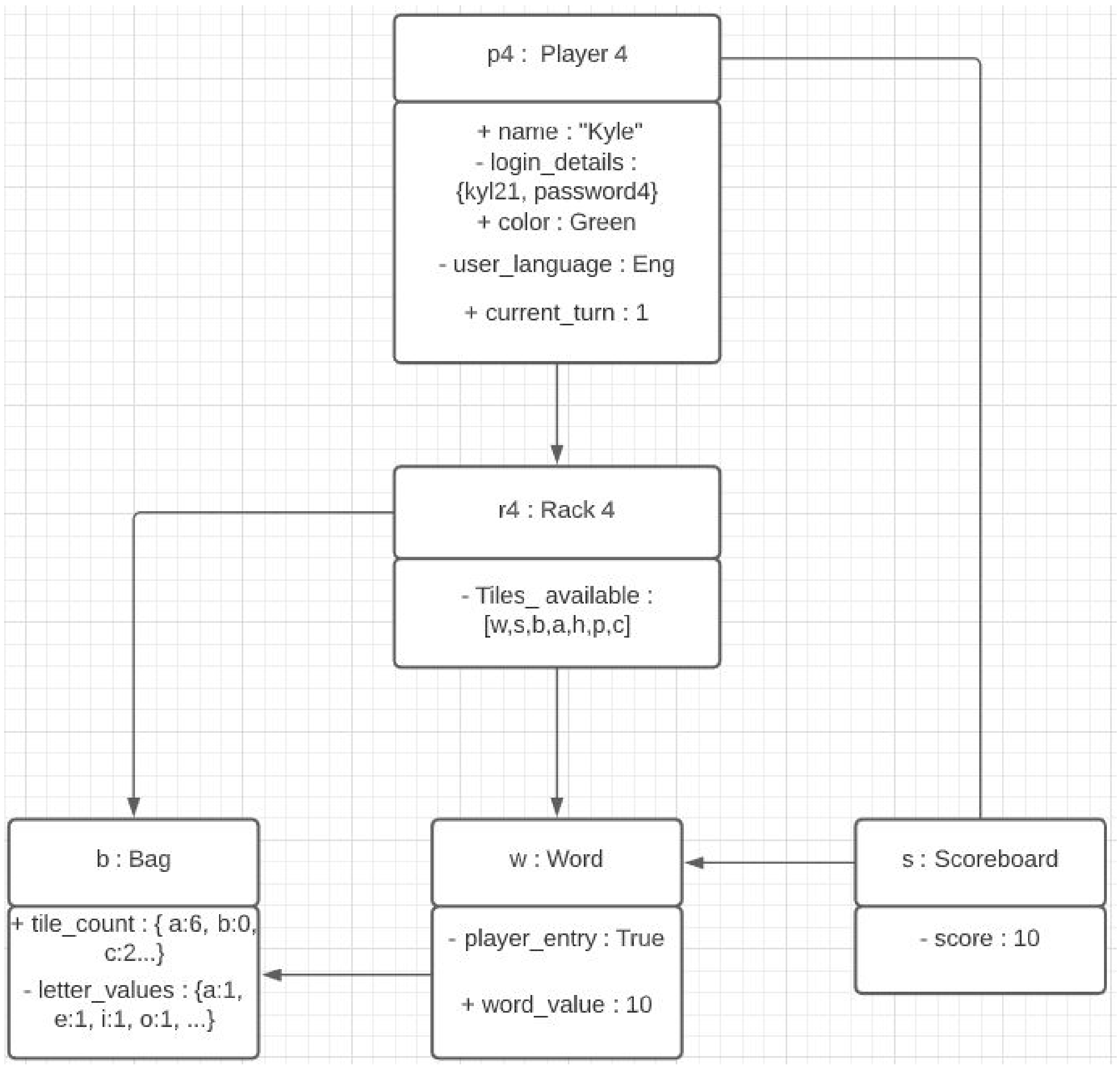


6.4 Local Diagram

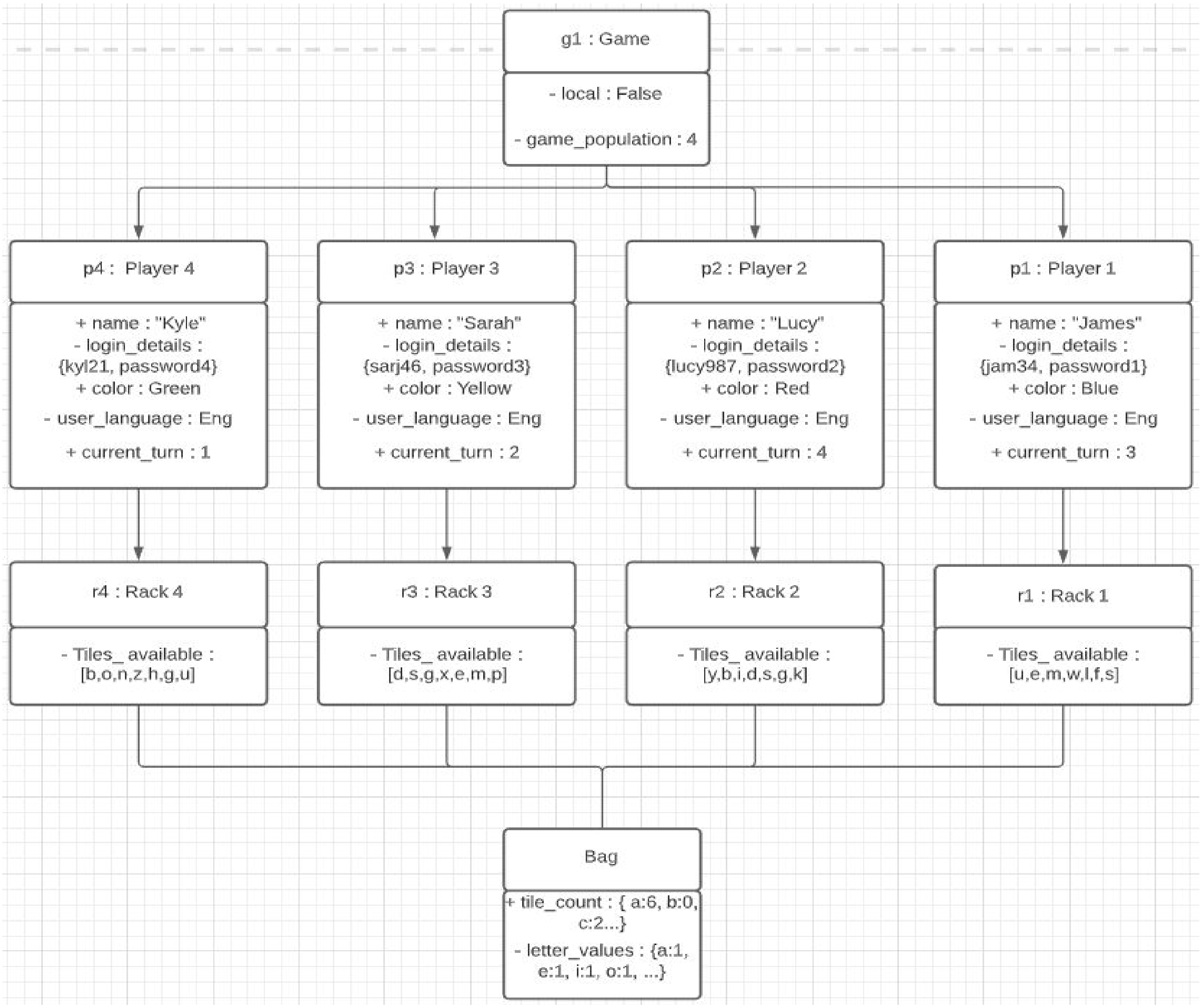


7. Revised Object Diagram

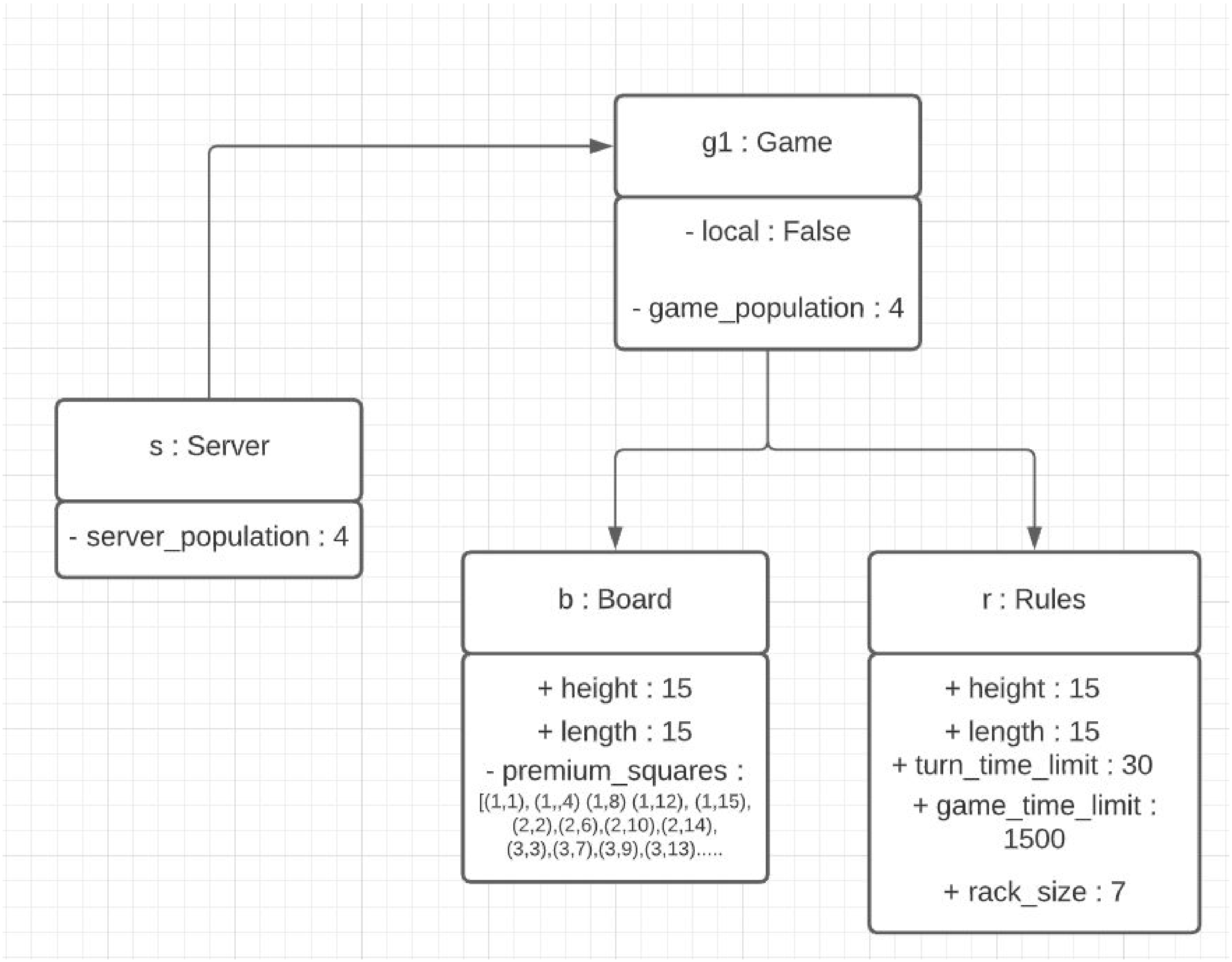
7.1 Player plays a word



7.2 Setting up players and racks

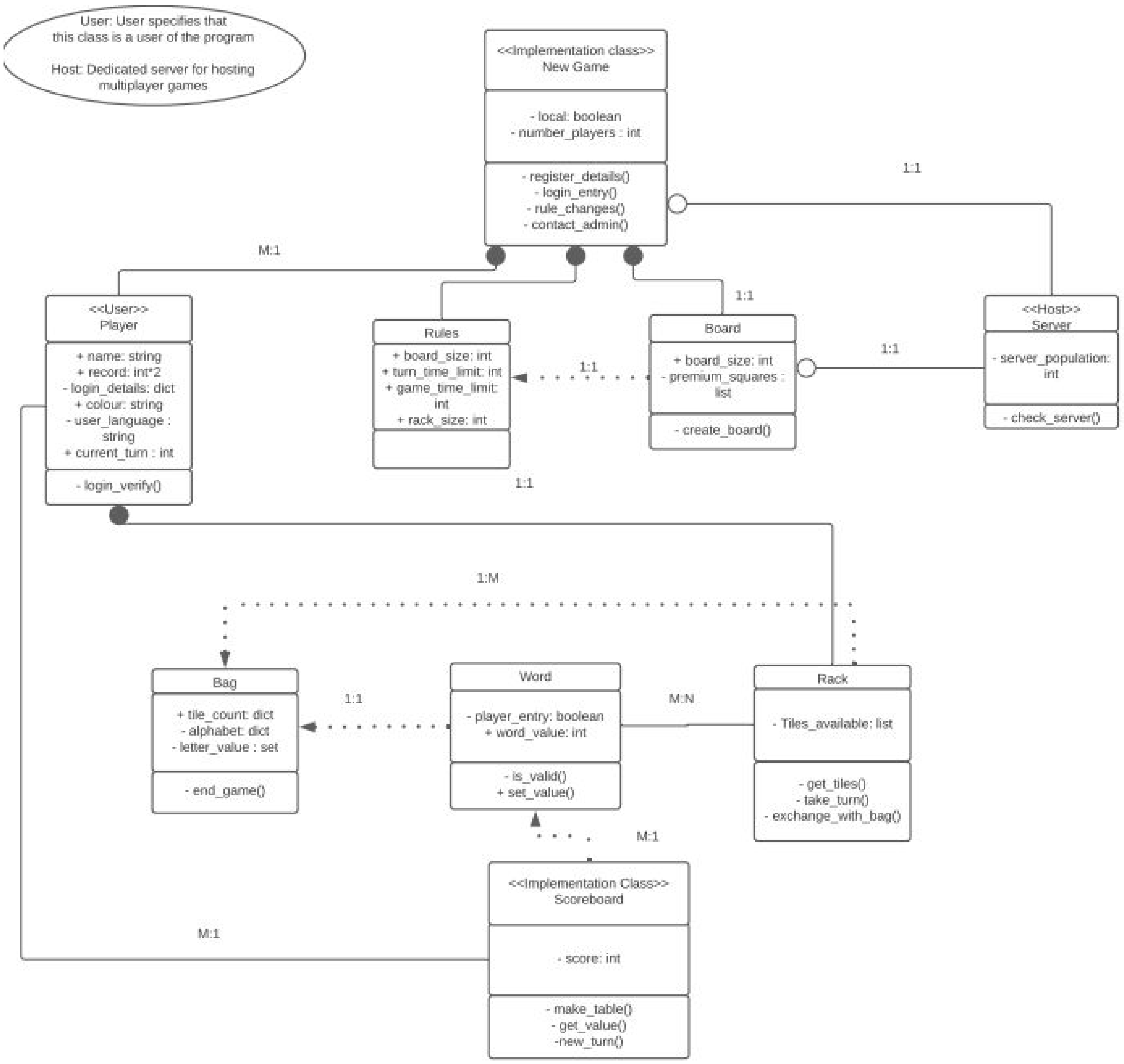


7.3 Online Game Initialization



****Attribute visibility indicators added to all diagrams****

8. Revised Class Diagram



Class Skeleton

```
Class New_Game():
    Boolean local
    Int number_of_players

    def register_details():
        #set players name and password if applicable

    def login_entry():
        #only called if online game
                #waits for name and password of player for remote game

    def rule_changes():
        #sets rules for game//set values in Rules class

    def contact_admin():
        #allows user to report bugs or errors in game

Class Player():
    String Name
    Int record
    Dictionary login_details
    String colour
    String user_language
    Int current_turn

    def login_verify():
        #checks if username and password match for given player

Class Rules():
    Int board_size
    Int turn_time_limit
    Int game_time_limit
    Int rack_size

Class Board():
    Int board_size
    List remium_squares

    def create_board():
        #creates a grid based on rules set (board_size value)

Class Server():
    Integer server_population

    Def check_server():
        # Hosts online game
        # Updates the board and scores for all players
```

```
Class Bag():
    Dictionary tile_count
    Alphabet dictionary
    Letter_value set

    Def end_game():
        # finishes the game once there are no more letters in the tile_count

Class Word():
    Boolean player_entry
    Integer word_value

    Def is_valid():
        # checks if the word played by the player is valid

    Def set_value():
        # calculates the value of the word (if it's valid)

Class Rack:
    List tiles_available

    Def get_tiles():
        # Take tiles from the bag
        # First time - take seven letters
        # Otherwise - only take as many have been played

    Def take_turn():
        # Allow player to play a word

    Def exchange_with_bag():
        # Allow player to swap a certain amount of tiles between their bag and rack
        # This ends their turn

Class Scoreboard():
    Integer score

    Def make_table():
        # Make table to be seen in game

    Def get_value():
        # Gets value from Word.set_value()
        # Adds value to the players score

    Def new_turn():
        # Starts the next players turn
```

10. Group Minutes and Agenda

Meeting 1: Here We Go Again
Date: 04/11/2020

- Discussed the next set of tasks in this deliverable.
- Began to research the next set of tasks.
- We discussed what challenges we might face within this workset.
- Set our next meeting for the 10/11

Meeting 2: Getting the Ball rolling
Date: 10/11/2020

- Discuss the issues we have with what we have created thus far.
- Brought up any feedback we had on completed work.
- We changed what was needed based on the feedback given.
- Set the next meeting 13/11

Meeting 3: Tipping away
Date: 13/11/2020

- We have finished the first draft of our tasks and reviewed them as a group to refine them to be delivered.
- Applying the finishing touches on what has been completed thus far.
- Set a meeting to add everything together to be delivered.

Meeting 4: And off we go
Date: 20/11/2020

- Added all our work into one folder ready for submission.
- Added the finishing touches to our deliverable.
- Discussed the implementation in our next set of tasks.