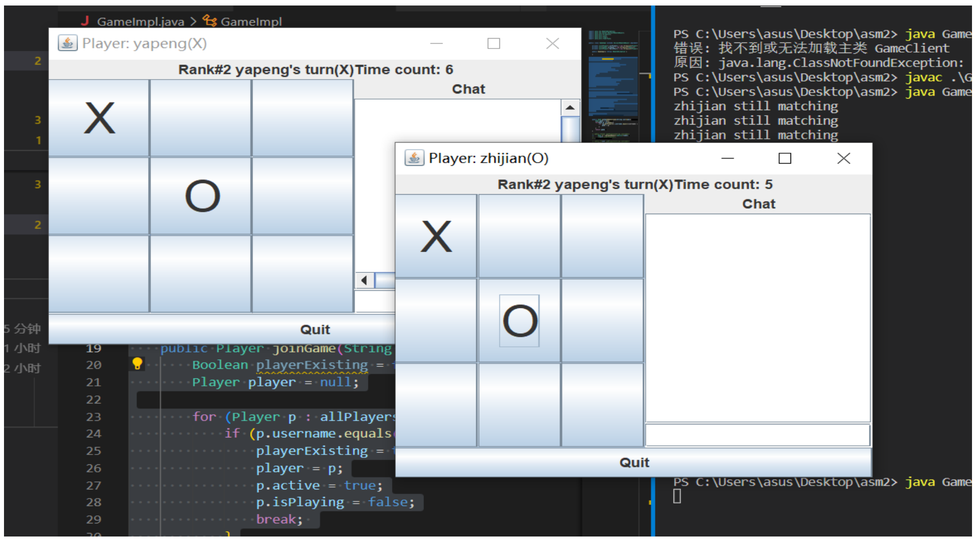
Name:zhijian yang
Student ID:1426234

1.The problem context

In this project,it's about creating a multiplayer Tic-Tac-Toe game using Java RMI (Remote Method Invocation). The game will allow multiple players to connect, play, and chat in real-time. We'll also add some extra features, like handling server issues, player time limits, player rankings, and more.

Game Rule

In Tic-Tac-Toe, two players take turns marking a spot on a 3x3 grid with 'X' or 'O'. The goal is to get three of your marks in a row, either horizontally, vertically, or diagonally. If all spots are filledand no one wins, the game ends in a draw.



```
    public Player joinGame(String
        Boolean playerExisting = 
        Player player = null;

        for (Player p : allPlayers
            if (p.username.equals
                playerExisting = 
                player = p;
                p.active = true;
                p.isPlaying = false;
                break;
```

```
PS C:\Users\asus\Desktop\asm2> java Game
错误: 找不到或无法加载主类 GameClient
原因: java.lang.ClassNotFoundException:
PS C:\Users\asus\Desktop\asm2> javac .\G
PS C:\Users\asus\Desktop\asm2> java Game
zhijian still matching
zhijian still matching
zhijian still matching
```

```
PS C:\Users\asus\Desktop\asm2> java Game
```

[The screenshot of game Interface]

Basic System (Client)

The player's side of the game will have these basic parts:

- Player Setup: Players choose a unique username when they start the game. We assume everyone will have a different username.
- Visual Interface: The game displays a simple graphical board so players can see the game. It also shows who's playing and their symbol ('X' or 'O').
- Finding a Game: When players start, they'll see "Finding Player" until a match is found. Players are ready as soon as they start the game.
- Taking Turns: During a game, the player's name and their symbol are shown. To make a move, they click on an empty spot.

- Live Updates: As players make moves, both players' boards are updated. The game sends move information to the server.
- Chat: There's a chat window for players to talk during the game. It keeps the last ten messages and shows the newest at the bottom.
- Leaving the Game: There's an 'exit' option so players can leave at any time.
- Game Outcome: After the game, it says who won or that it was a draw. Players can start a new game or leave.

Basic System (Server)

On the server side, we need to do the following:

- Handling Many Players: The server can deal with lots of players who want to play.
- Finding Opponents: The server matches two players to start a new game. It keeps matching as long as there are players.
- Assign Player Mark: The server randomly picks who goes first and who gets 'X' or 'O'.
- Game Updates: The server keeps the game updated as players make moves and tells the players what's happening.
- Winning Checks: After every move, the server checks if someone has won. When the game ends, it tells both players.
- Player Leaving: If a player leaves the game early, the other player is the winner.

Extra Features

In addition to the basics, we've added some extra stuff:

- Dealing with Server Issues: When server crashes, all players will see tips and they will force logout from the interface after 5s.
- Player Time Limits: To keep the game going, there's a time limit for each player's turn. If they do not make a move until time out,a random move will be made, and the other player gets a turn.
- Player Rankings: We keep track of player rankings based on their history match.Win or lose will earn or lose 5 points, and earn 0 point if the game is a draw.

2.Description of the components of the system

2.1Game Logic

Player

- The Player class represents individual players within the game.
- Each player has a username to identify them, a mark (either "X" or "O") to indicate their symbol in the game, points to keep track of their score, and boolean flags for isPlaying (to indicate if they are currently in a game) and active (to show if they are online and available to play).

Game

- The Game class represents a specific game of Tic-Tac-Toe being played by two players.It contains all core elements and basic logic of a Tic-Tac-Toe game.
- Player Interaction: Implement features to facilitate player engagement, including make a move, result anticipation, chat tracking, and game muting.
- Game state: It maintains the game state, including the game board, the players involved (playerX and playerO), and the current player's turn.The server can retrieve game state to obtain up to date game information.

2.2 Client Interface Implementation

GameClient

- This is the entry point for the client's startup. It will create a game interface for the user, inform the server that it has joined the matchmaking queue, periodically check with the server to see if a game has been created for the user (meaning a match has been found), and if so, update the server's game status on the interface.
- When game players need to interact with the server, such as performing actions, exiting, searching for new matches, and so on, the game client calls remote server methods to update the game state.

2.3 Remote function component

GameServer

- This is the entry point for the server's startup, registering all the methods in GameInterface used for client side to access
- Inform all players when the server crashes.

GameInterface

- This is a JAVA interface that lists all the methods for playing games remotely. It aims to improve gameplay by making it easier for players to interact with the game and each other from different locations. These methods mainly include tasks like managing players, checking the game's status, enabling chat, making moves, and other important game operations.

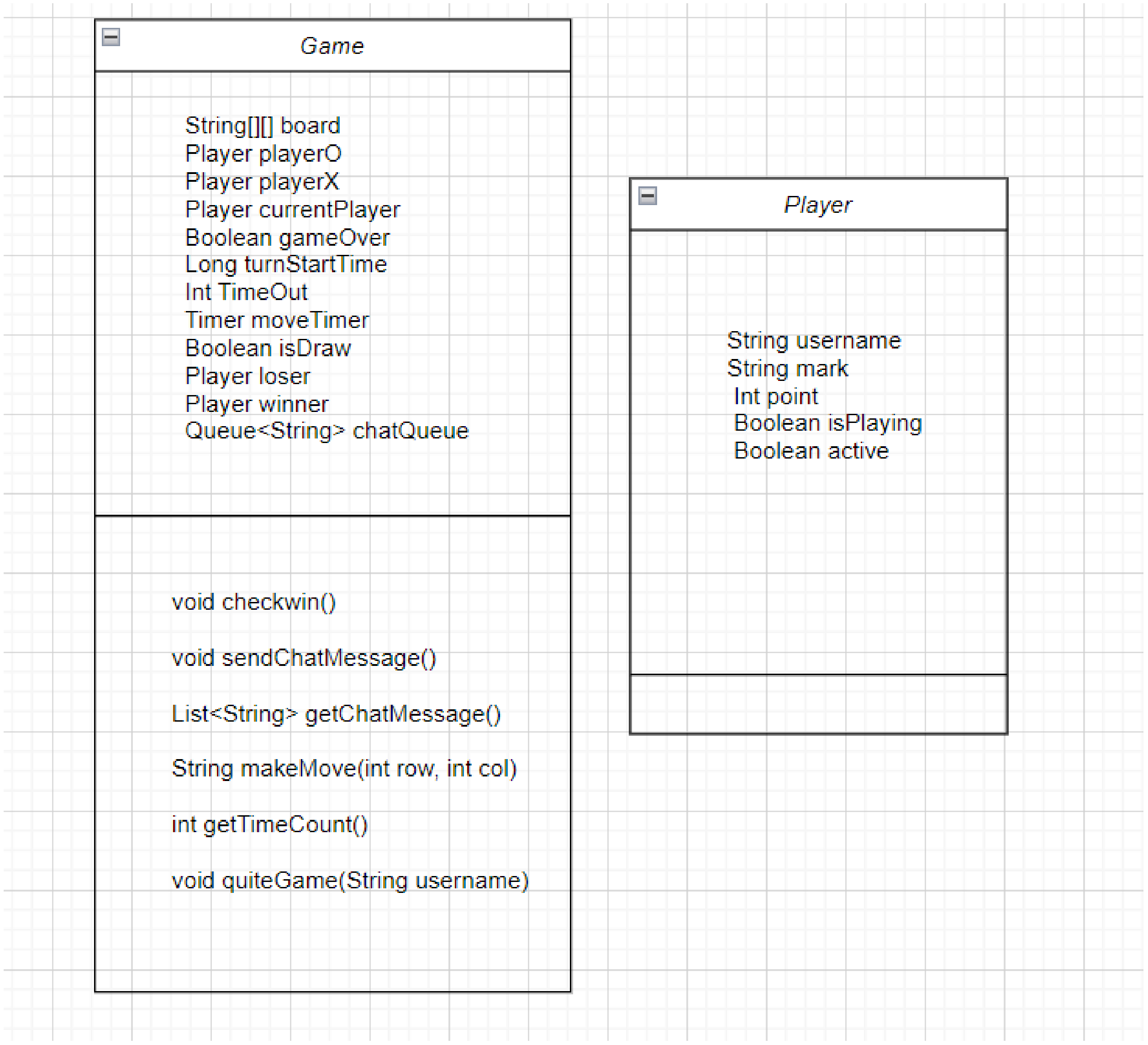
GameImpl

- It is an implementation of GameInterface that serves to store games and player information. It plays a crucial role in managing client requests by identifying the relevant game through matching the provided username.
- Player Join Game Request: move the player to the active player pool. If the number of players in the pool is even, the system will automatically create a game and store it.
- Player Interaction Request: Find its game and call the inform game.
- Retrieve Game state: Find its game and return its state.
- Player Management: Store and maintain records of all players, including their status, online/offline presence, in-game activities, and historical match points earned.

3.Overall class design

Game Logic Class

Game && Player



Key Function

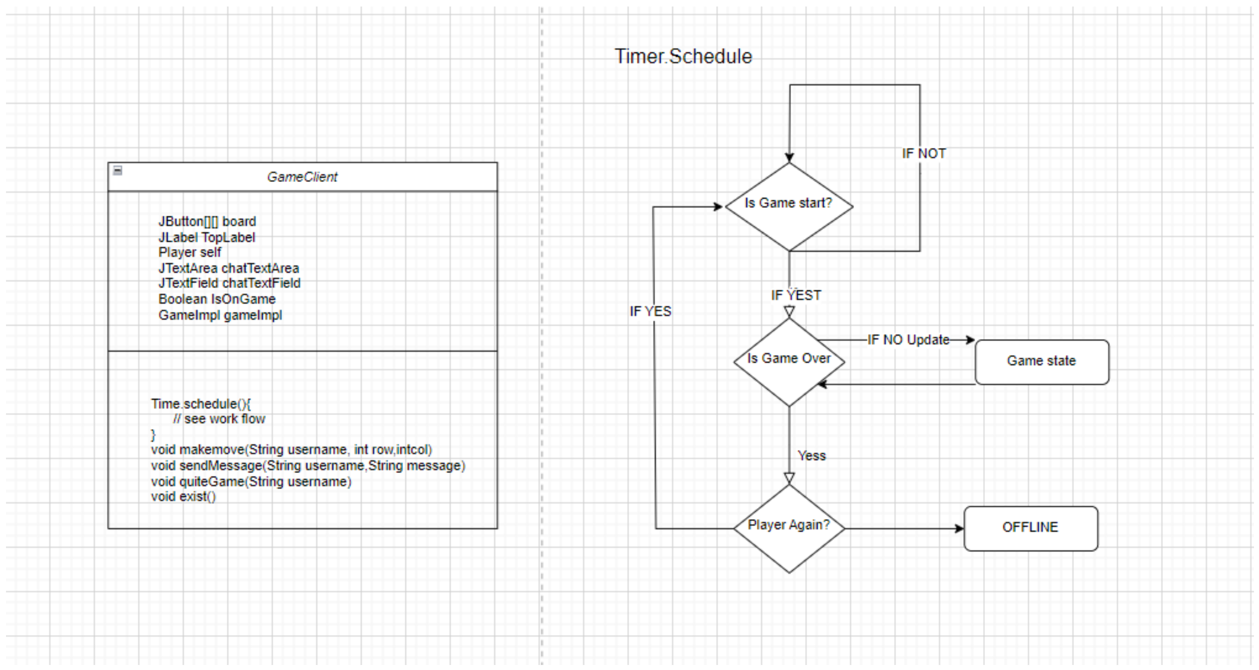
- Checkwin: It checks if there are three consecutive "X" or "O" marks in rows, columns, or diagonals and assigns a winner and loser accordingly.

- makeMove:It updates the board with the player's move, switches to the other player's turn, and sets a timer to limit how long a player has to make their move. If a player takes too long, the timer automatically makes a random move for them. If the move is valid, the function returns "success move."

- quiteGame:Find a player using their username and say they lost the game, and the other player won. Then, make it so both players can't play anymore, and the game is over.
- getTimeCount: Return the time left in this player's turn
- get/sendMessage: Add/Query the message in chatQueue.

Client Side Class

GameClient



```
JButton[][] board
JLabel TopLabel
Player self
JTextArea chatTextArea
JTextField chatTextField
Boolean IsOnGame
GameImpl gameImpl
```

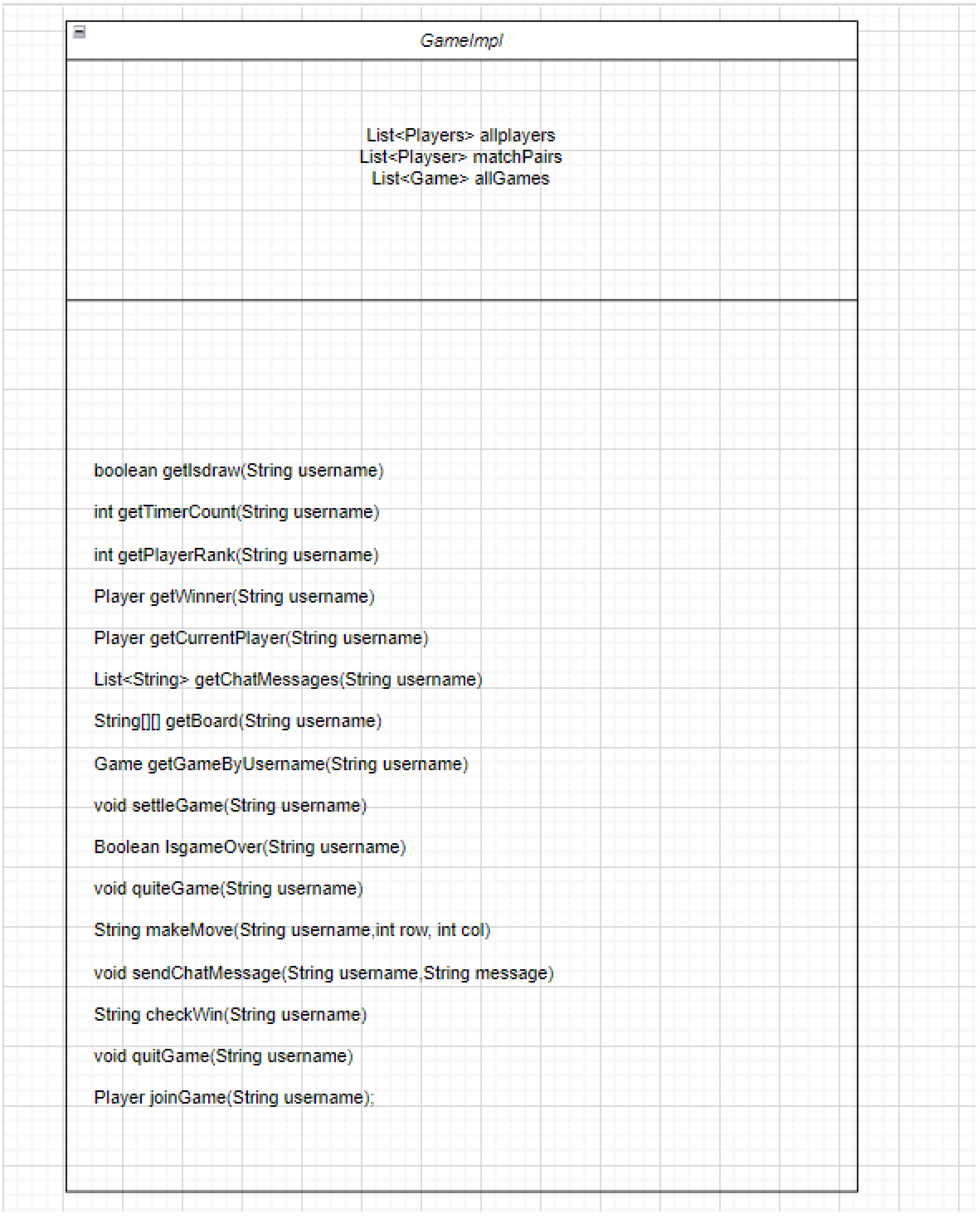
```
Time.schedule(){
     // see work flow
}
void makemove(String username, int row,intcol)
void sendMessage(String username,String message)
void quiteGame(String username)
void exist()
```

Key Function

- Time.schedule: It is kept running until the client exists, it mainly asks the server if the player matches the game and if so updates the Game state from server to interface.
- makeMove:The username is used for the server to find its game.When Player draws on the board on the interface, it will update to the server board.
- quiteGame:Tell server player is quiet since this player click quit button
- sendMessage:Tell server this player is sending message since player typing message in chat box
- Exist: Exist the interface windows when server crashes

Remote Function

GameImpl

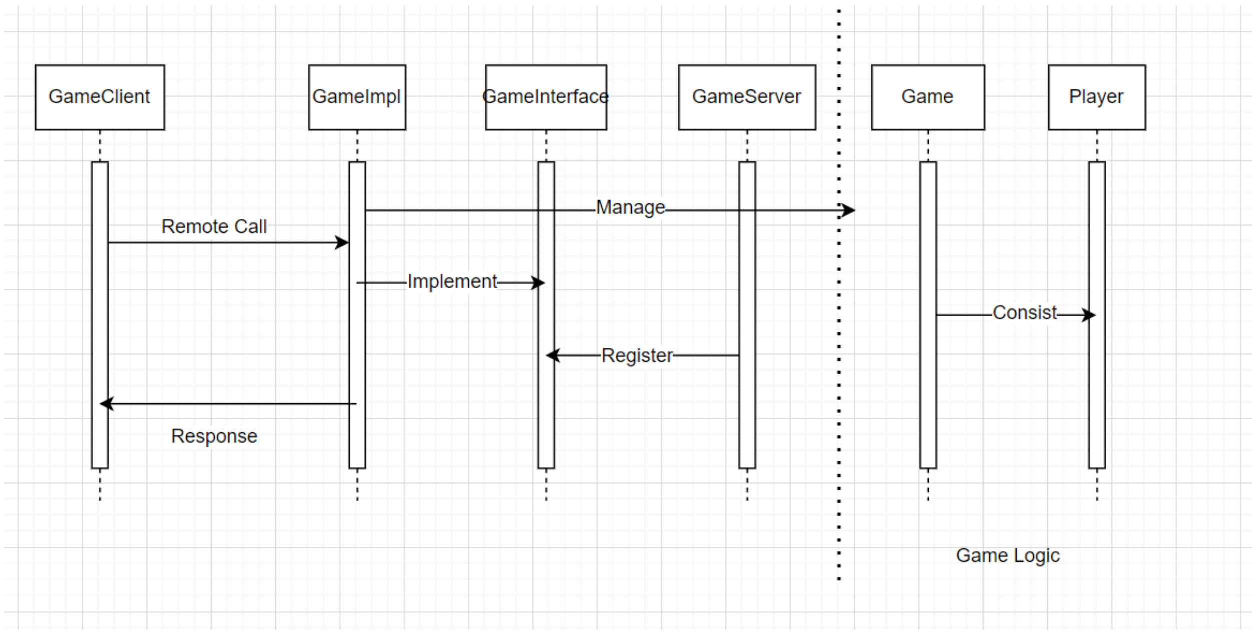

GameImpl
List<Players> allplayers
List<Playser> matchPairs
List<Game> allGames
boolean getIsdraw(String username)
int getTimerCount(String username)
int getPlayerRank(String username)
Player getWinner(String username)
Player getCurrentPlayer(String username)
List<String> getChatMessages(String username)
String[][] getBoard(String username)
Game getGameByUsername(String username)
void settleGame(String username)
Boolean IsgameOver(String username)
void quiteGame(String username)
String makeMove(String username,int row, int col)
void sendChatMessage(String username,String message)
String checkWin(String username)
void quitGame(String username)
Player joinGame(String username);

Key Function

- All the function with the prefix with get is used to obtain game state, the gameImpl finds the game by username and returns its related state.Some function is used for player behavior, etc makeMove, for those functions, gameImpl just calls the game's inside method to update its game state.

- joinGame: This function either adds a player to the "MatchPairs" list or creates a new player if they are joining for the first time. When there are two players in "MatchPairs," it initiates a new game for them.
- settleGame: Settle the player's game score and remove this game from allGames because the game has already been completed.
- getPlayerRank: Sort the players by their point and return the index of player

Interaction diagram.



4.Critical analysis and conclusion

The client interface doesn't require an in-depth understanding of the game; it functions more like a front-end and relies on the server to provide it with the latest game status. The game's various components are structured in a way that resembles different classes, making it a wise choice to implement object-oriented programming. The fundamental rules of the game and how players interact with it are all managed within these classes. Both the client's experience and the game state for individual players are overseen by a central game class.

However, when we seek to manage multiple players and games, a server becomes a necessity to orchestrate and monitor all these aspects efficiently. By storing all game-related information on the server, we open up opportunities for various functionalities, including creating player rankings, tracking match histories, assisting clients in reconnecting to their games, and implementing game pausing. These advantages underscore the primary benefits of adopting a server-based system.

With numerous games in play, it's essential to be able to identify the specific game a client is involved in when their requests are received. This is typically achieved by associating a client with a particular game based on their username. This process ensures that the game's status on the server is always up to date. So, the next time the client requests information about the game, they get the most current data.

Optimization is also a key consideration. Currently, each client frequently asks the server to update their game's status. In scenarios where there are many active users participating in the game, generating a high volume of requests to our server, this can strain the system. To address this concern, a technique called caching can be employed. The idea is to store the information most frequently requested by clients in memory. When a client asks for this data again, we can swiftly provide it, reducing the server's workload and ensuring faster response times, ultimately enhancing the user experience.