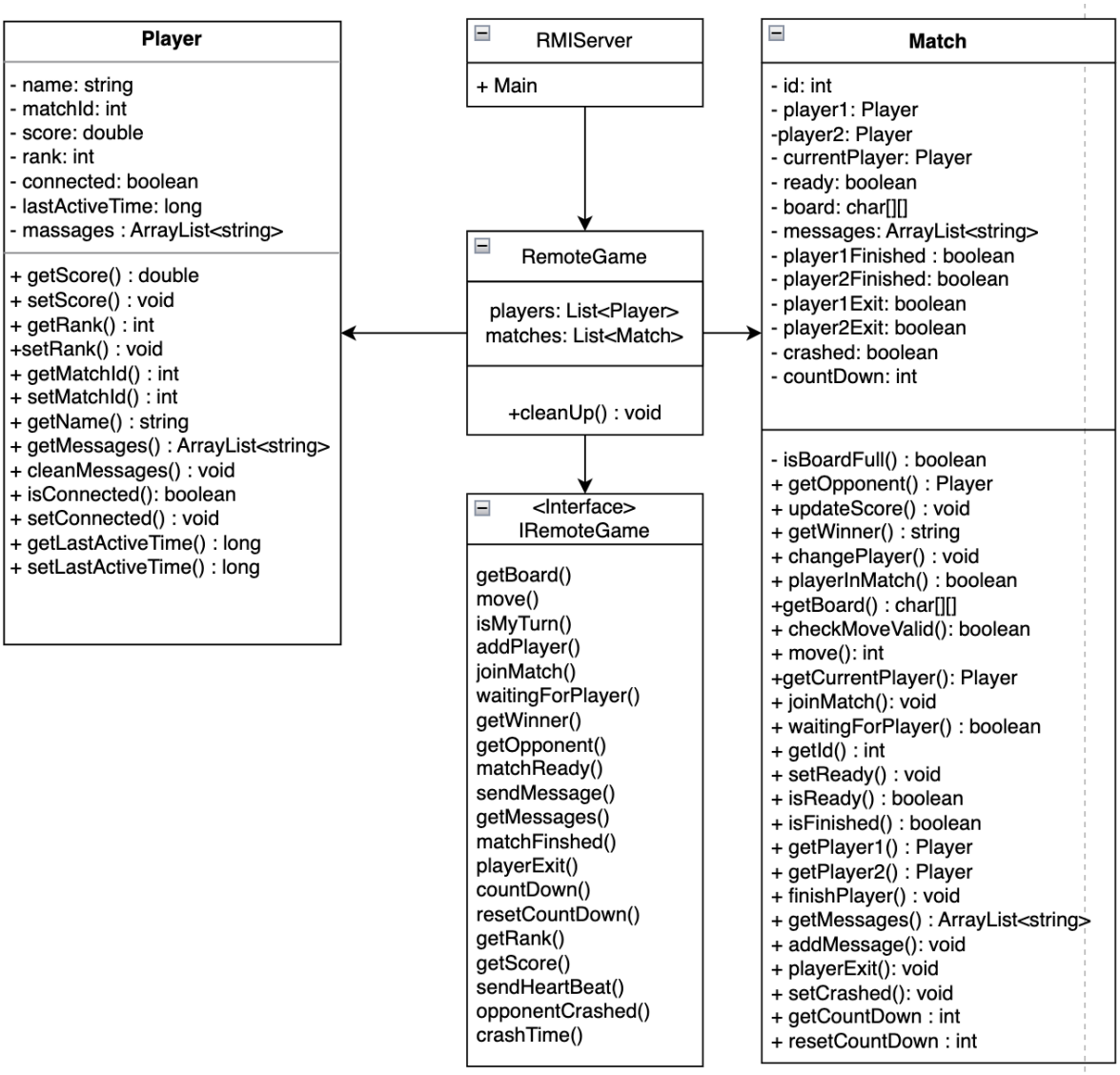
COMP90015 Assingment 2

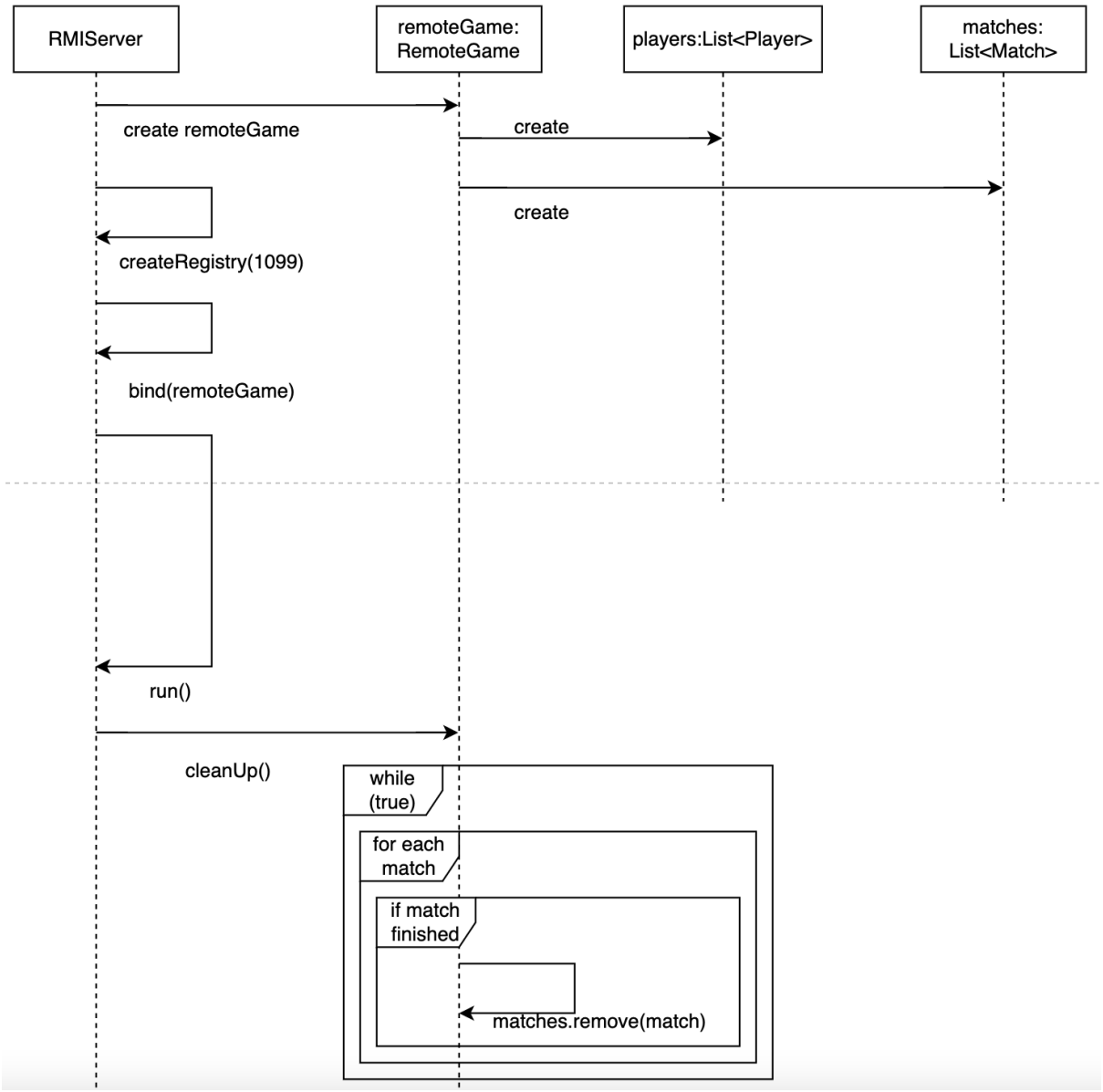
Zongliangh Han
1166050

For this assignment, the task involves creating a Tic Tac Toe game application. This application is separated into 2 parts, client and server, in order to meet the criteria of assignment. Remote Method Invocation (RMI) is used to implement the project to support the functionalities of playing tic tac toe game with other users, getting countdown, random move after timeout, fault tolerance when client or server crashed. User's score and rank will also be rendered on the board.

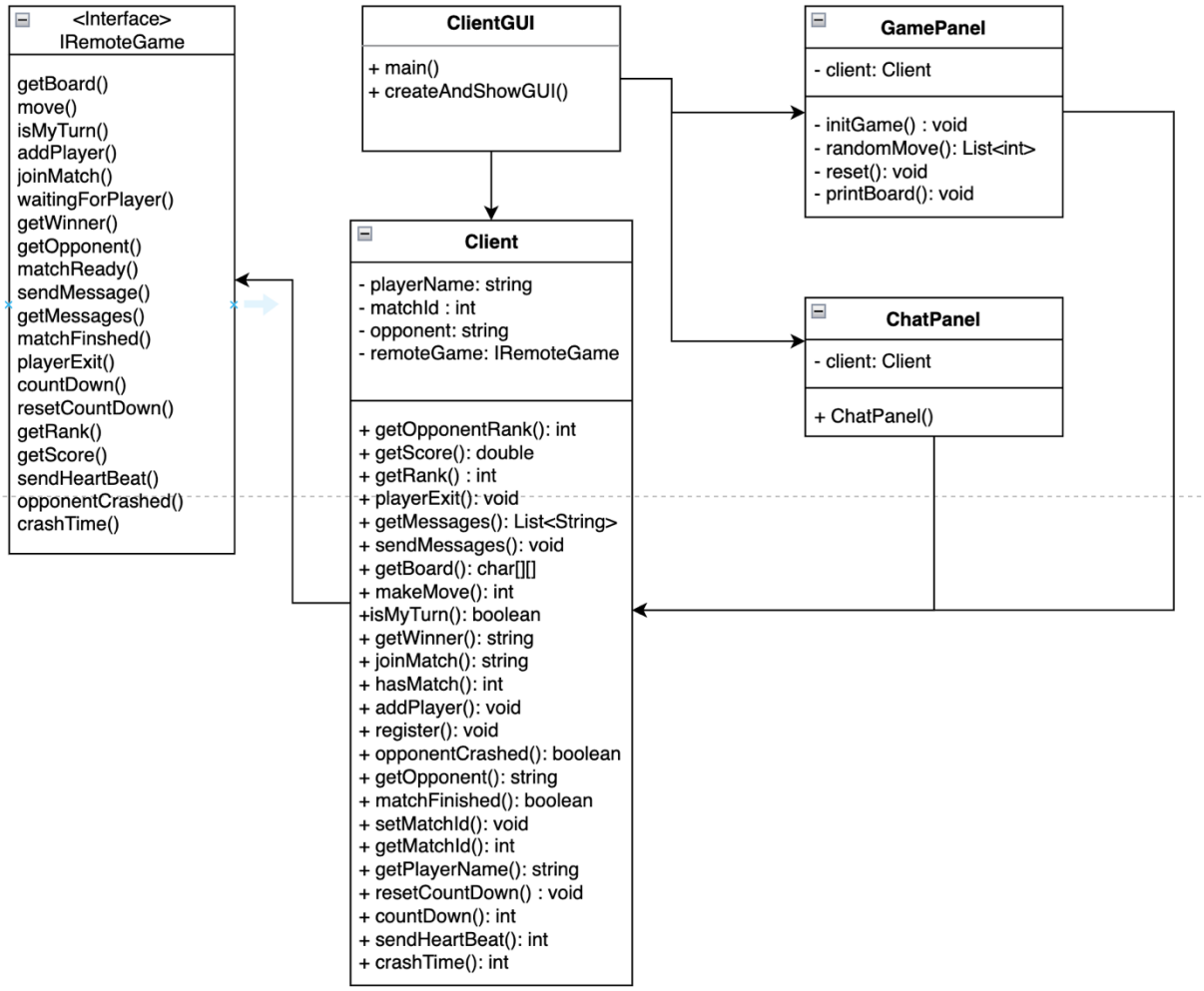
Server application



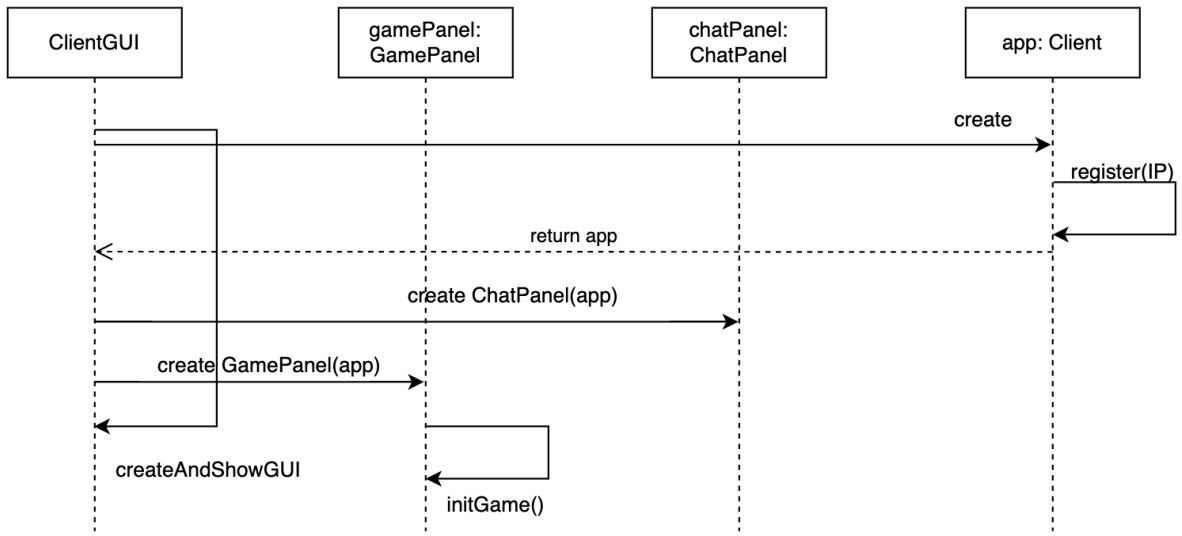
RemoteGame class implemented the RMI methods. It also keeps track of current players and ongoing matches. RMI server create the RemoteGame object and continuously remove finished matches from list.



Client Application

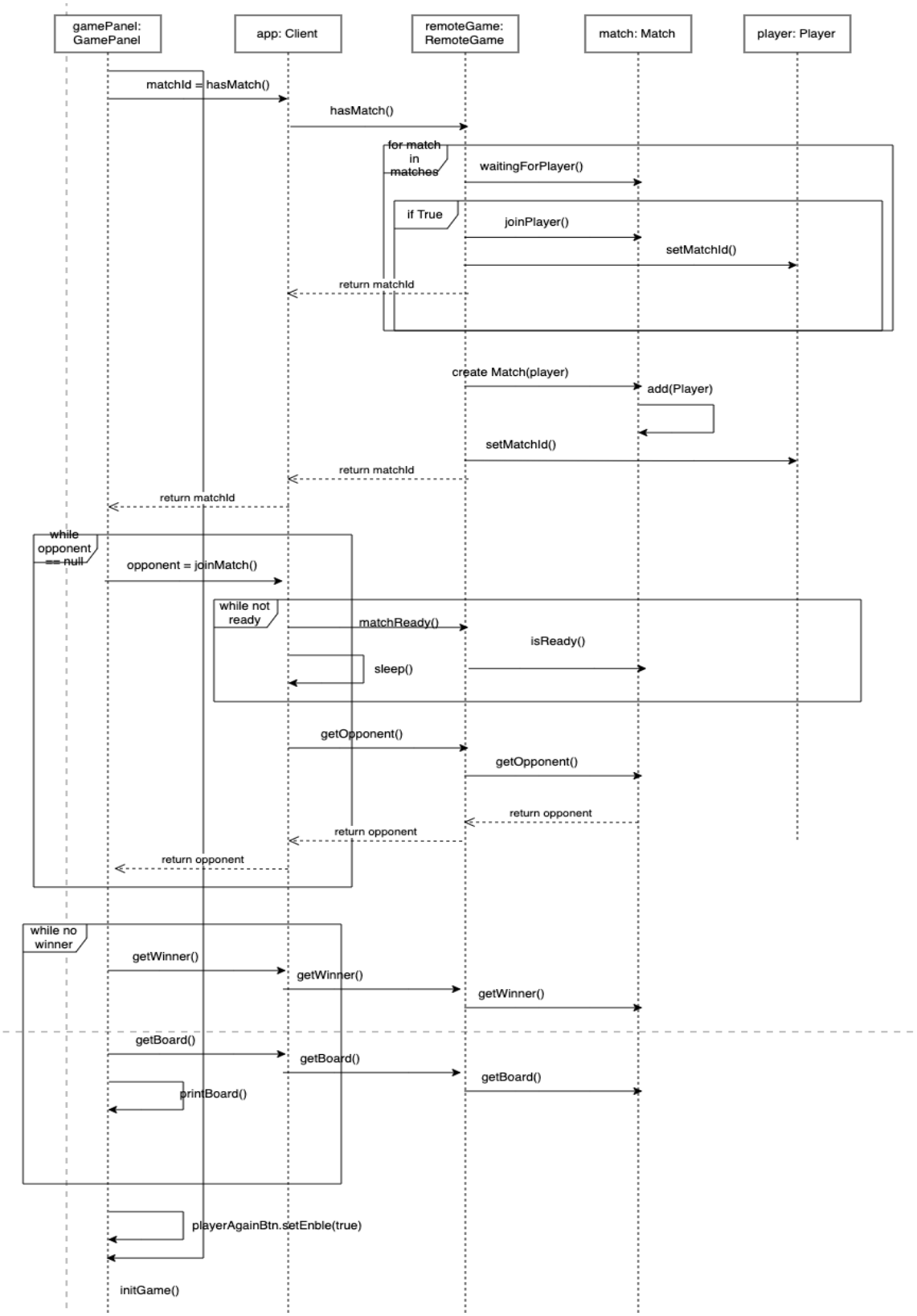


ClientGUI will use createAndShowGUI() method to create GamePanel and ChatPanel. A Client instance will also be created and passed to both GamePanel and ChatPanel. Each user will be corresponding to one Client instance, which contain the important information of user, such as player's name, match id and opponent name. The method in client class calls the RMI method in IRemoteGame interface. ChatPanel and GamePanel is able to use RMI method through Client instance.



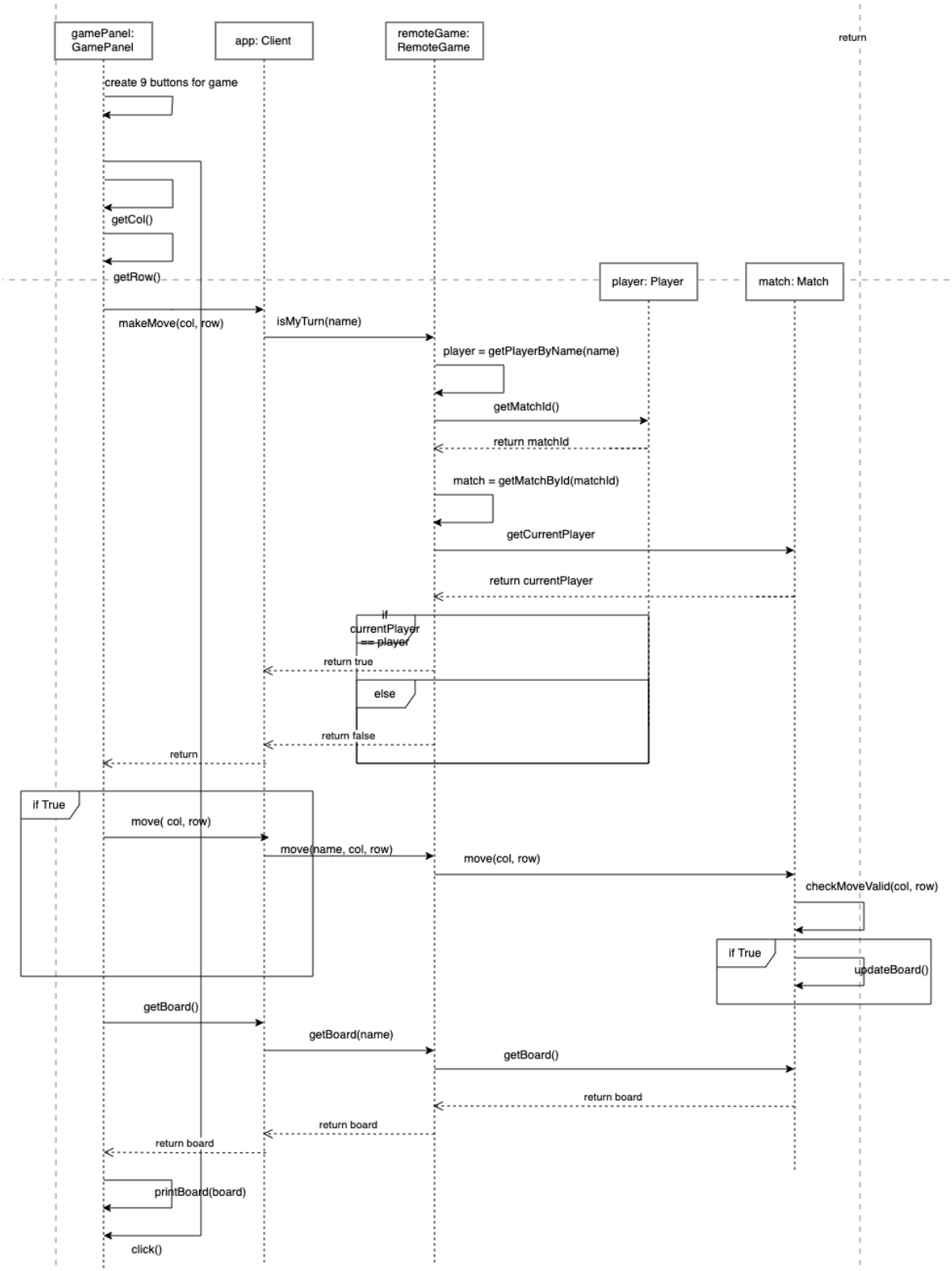
Key functionalities

1. Start Game:



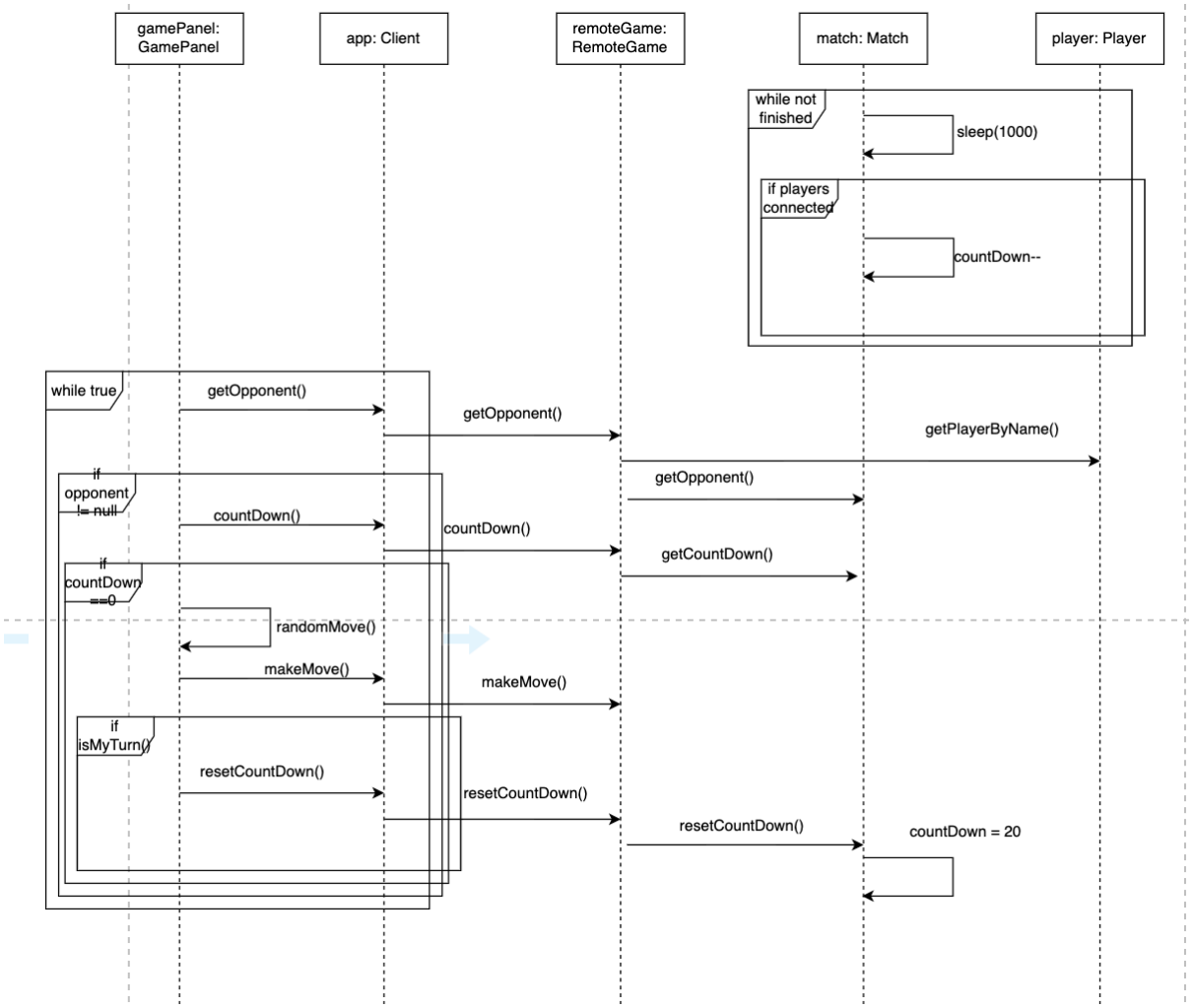
When client create gamePanel, it will try to check if there is any match available to join, if not it will create a new match and continuously wait for a new player to join the match. When there are two players joined the match, it will start. GamePanel will keep listening to the player moving event until the game is finished.

2. Make a move



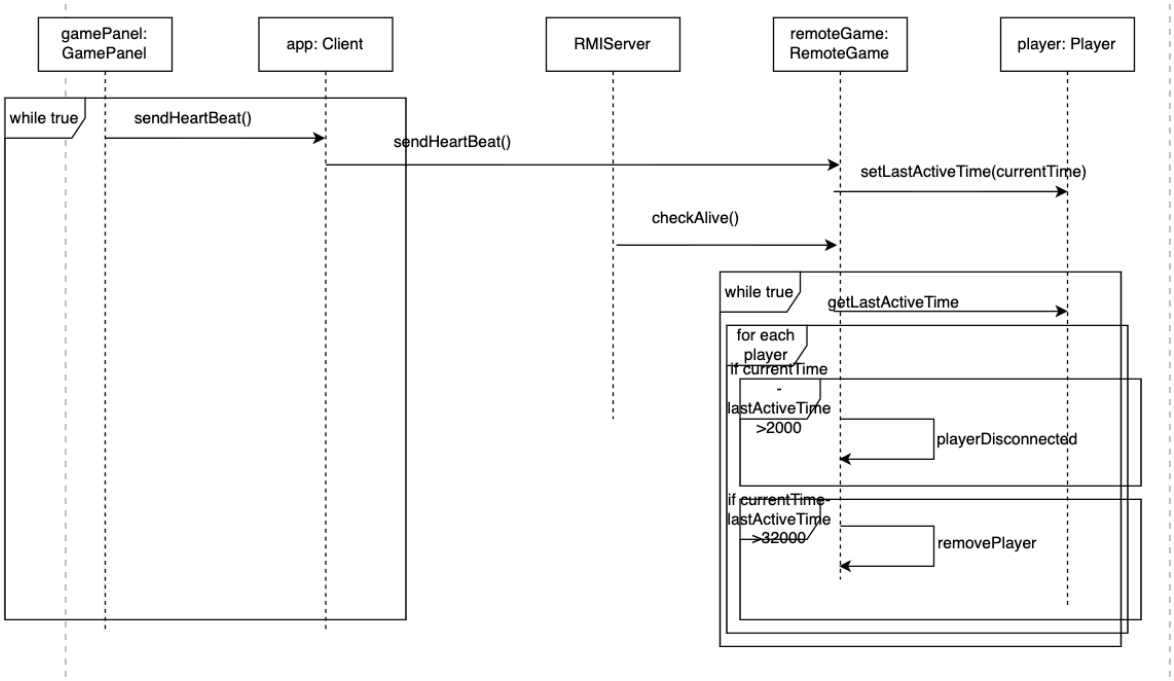
9 buttons will be created the as grid for player to make the move. After the player click the button, it will extract button's column number and row number, then send to server side. In the server, it will check if now is the player's turn and if the move is valid. If both true, the move will be made. GamePanel then will extract updated map from server.

3. Count Down



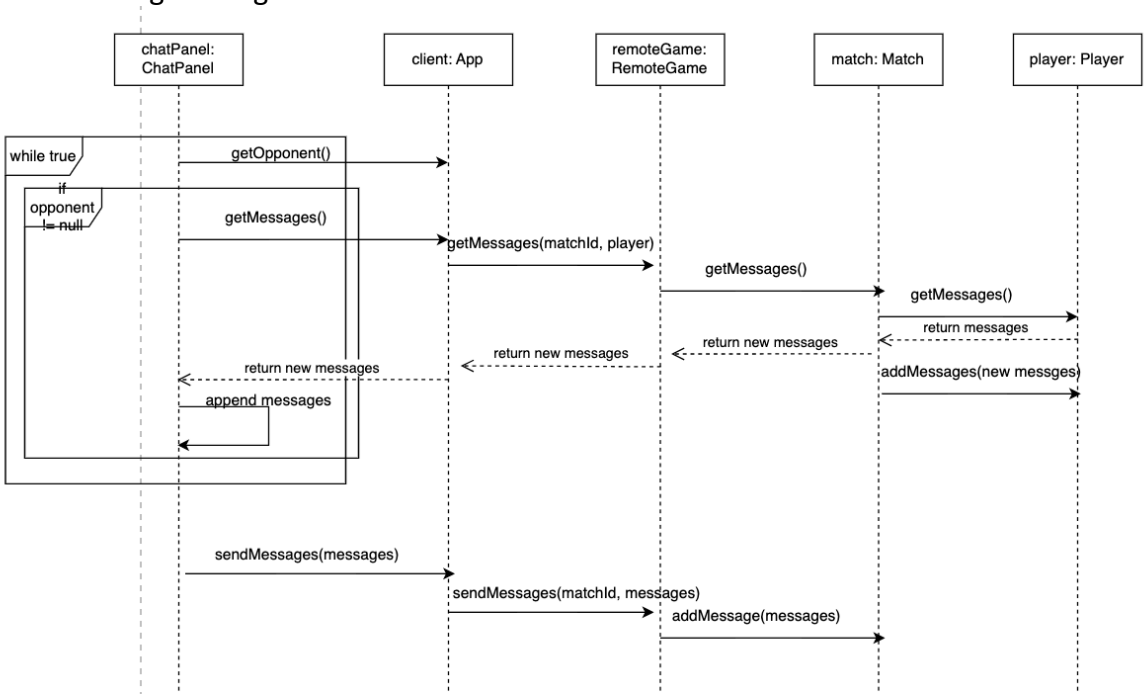
After match is ready and 2 players are both connected with server, match will start a new thread to count down from 20. GamePanel will get the count down number through RMI method countDown() by using its player name. If player do not make a move before the countdown is 0, GamePanel will automatically generate a random valid move. After player made the move, GamePanel will call the RMI method resetCountDown() back to 20.

4. Sending heartbeat to server.



In order to check if client is connected with server, a new thread is created to send heartbeat message to server every 500 ms. In the server side, if one client did not send heartbeat message for 2s, it will consider this client is disconnected from server. Server will stop the countdown of its match and sending a message to inform this player's opponent. After 30s, this player will be removed from the players list and match will be set as draw game.

5. Sending messages



A new thread is created to constantly checking if there are new messages arrived. If there exist a message in match that is not in player, this message will be considered as new message and appended in chatPanel. When player try to send message, the message will be added into match through sendMessege() method.

Database

In this assginment, a JSON file was used as mock database to store the relative information of user. It follow the format below:


```
[
  {
    "username" : "Harry",
    "score" : 12.0,
    "rank" : 1,
    "matchId" : 680,
    "messages" : [ ]
  },
  {
    "username" : "Larry",
    "score" : 7.0,
    "rank" : 2,
    "matchId" : 75,
    "messages" : [ ]
  }
]
```


The Player class has been designed as a model to facilitate the extraction of data from JSON files by using the jackson library. It also enables the modification of player's score, rank and matchId.

Analysis & conclusion

Both server and client application are implemented under the principle of low coupling and high cohesion. Tasks are divided into different classes in order to sepereate responsiblities. However, to be notice, GamePanel still contain game play, sending heartbeat, getting countdown, which can be further separated to different classes.

In order to handle concurrent issues, Match class contains additional variables to check the status of both players. All relevant status of match only is updated after both player's status changes. However, this approach introduces the complex in the system and slow down the performance of application. Further investigation can be made to coordinate the multiclient and server cooperation.

Furthermore, a real database can be made to store the player information. This will allow application to store large volume of data and speed up the data extraction rate, which will improve the application performance. In addition, since the high cohesion low coupling

design of system, this change can only made in GamePanel class without interfere with other classes.

In conclusion, the RMI-based Tic Tac Toe game demonstrates distributed gameplay while adhering to high cohesion low coupling design principles. Despite optimization potential, the system effectively handles concurrency in the Match class. Future enhancements may involve integrating a real database to improve data storage and retrieval efficiency. While there is room for improvement, the project successfully achieves its objectives and showcases potential for further refinement and scalability.