William Sinks
Radhesh Choudhary
Socket Full Project

Message format-

We used strings to send the command to the manager process. Once the string is received by the manager, the manager then checks the message for a specific command and executes the command included in the message. Each command is executed differently depending on the command.

Register: The manager splits the message up into a list of strings using “ “ as a delimiter and removes the command from the message. This is the player information syntax ⟨player⟩ ⟨IPv4-address⟩ ⟨m-port⟩ ⟨r-port⟩ ⟨p-port⟩. The manager then sends a Succeed or Failure message to the player in the form of a string.

Query Games- The manager receives the string message Query_Games from the player and then sends back a string including how many active games are ongoing as well as a list containing all the game information for each game.

Query Players- The manager receives the string message Query_Players from the player and then sends back a string containing how many players are registered as well as all Player information in a string.

De-register- The manager receives the string message De-register <Player name> and then sends back a Succeed or Failure string message back to the player.

Start-Game - The manager receives the string message Start_Game <Player name> <K> where K is the number of additional players to play the game with. The manager then sends back a Succeed (including the game identifier and game_info of all players) or a Failure string back to the player

End - The manager receives the string message End <Game-identifier> <Player name>. The manager then sends back a succeed (if the player name and game identifier matches information in the game info list) or a Failure string back to the player

Player Commands:
The following are the commands exchanged through the logical ring once the game has successfully started. This is done through the r port

Set-up - Once the dealer receives the success message from the manager, the dealer sends its neighbour a setup string message along with the game info list consisting of all the players currently in the game. This is circulated throughout the logical ring. Once the setup string has been received by a player, it keeps a copy of the game info, and based on the modulus function stores its neighbour's information

Deal - Based on the condition for the number of players playing, the dealer sends a deal string message to all the players playing the game along with their delt deck. Upon a receive of this message, the player keeps the deck as an information for the cards on their hand

Your-Move - This string message is sent to the neighbour of a player in the logical ring, indicating their turn.It also sends the remainder of the shuffled deck on the table to each player. When a player receives this message, it updates its information of the shuffled deck on the table, and proceeds its turn by checking its hand.If the hand and the shuffled deck on the table are not empty, it initiates the process of asking a random player for a card. If both are empty it skips its turn.

Update_Books - During their move, if a player is successfully able to make a book, it sends this string command along with its player index to the dealer, the dealer then checks if all the books have been won. If so, it sends a game over and a winner message. Otherwise it tells that player to continue its move.

Game-Over- This is a message sent by the dealer to the player who sent the update book command to indicate that all books in the game have been won, once the player receives this message, it prints out the books they have won.

Winner -

When a player apart from the dealer receives this string message, it sends the winner command to the dealer and also sends the books it has won. It also sends the winner command to its neighbour.

When the dealer receives the winner string message, it keeps track of the books sent to it in a rankings list. If it receives the number of books won from every player it finally prints out the winner and then sends the end command to the manager with its name and the game identifier

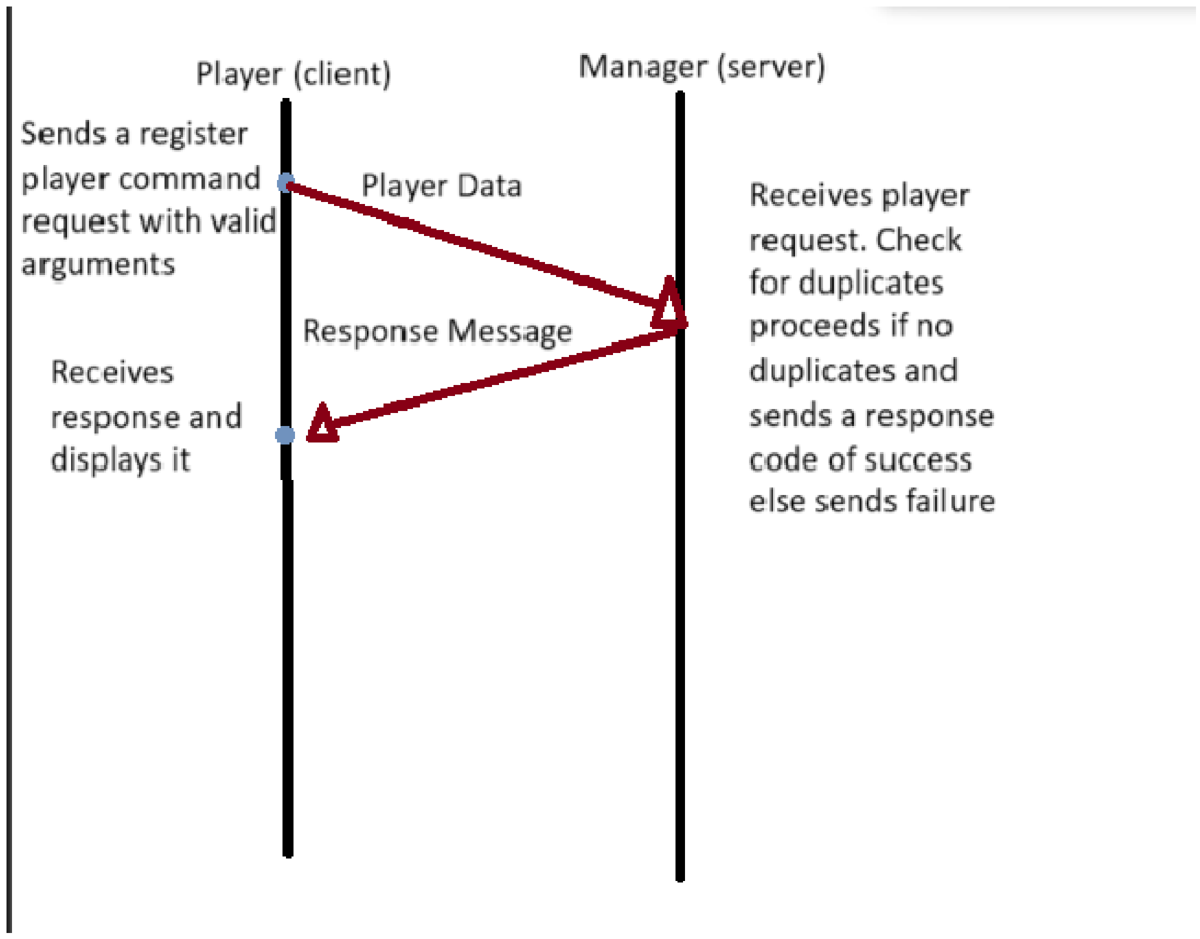
In our implementation the following is the messages exchanged by the peers during the game through the p port:

Ask -

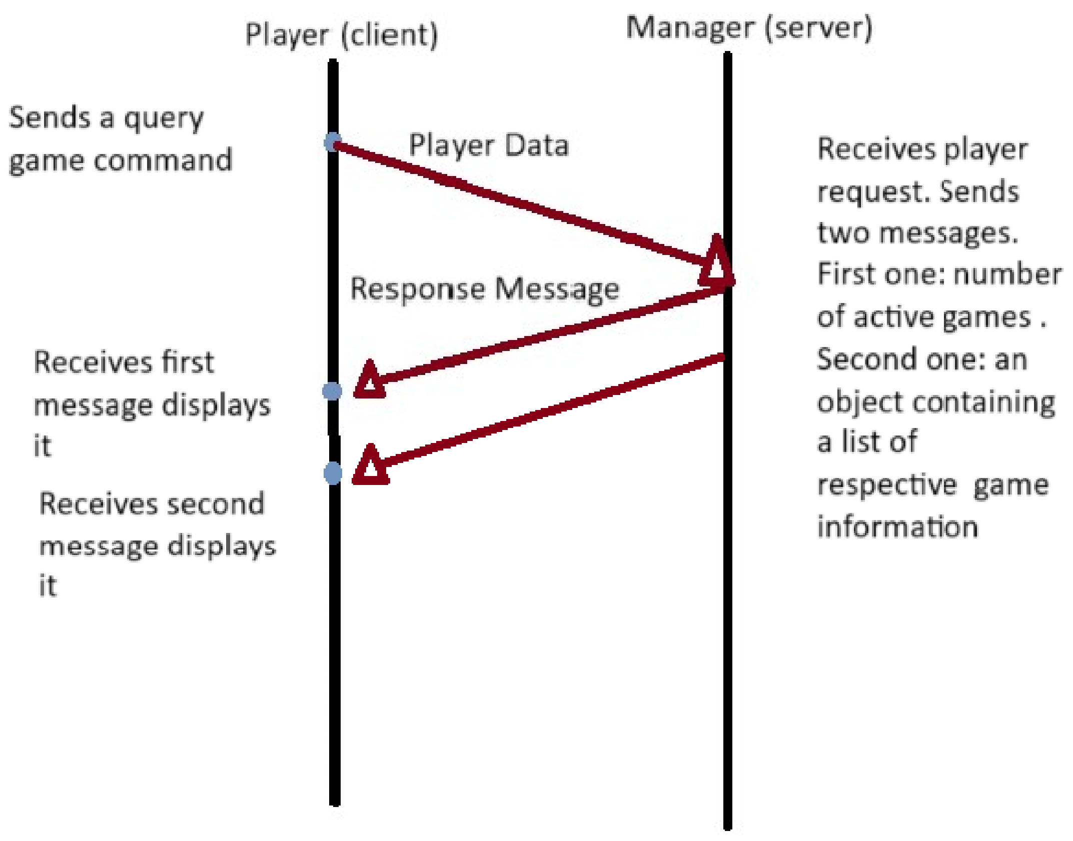
During a player's move, it sends this string message to a random player in the game along with the information for the card its asking (which is randomly generated based on its hand) The response string message is sent along with the count of the card that was initially asked for

Time- Space Diagram

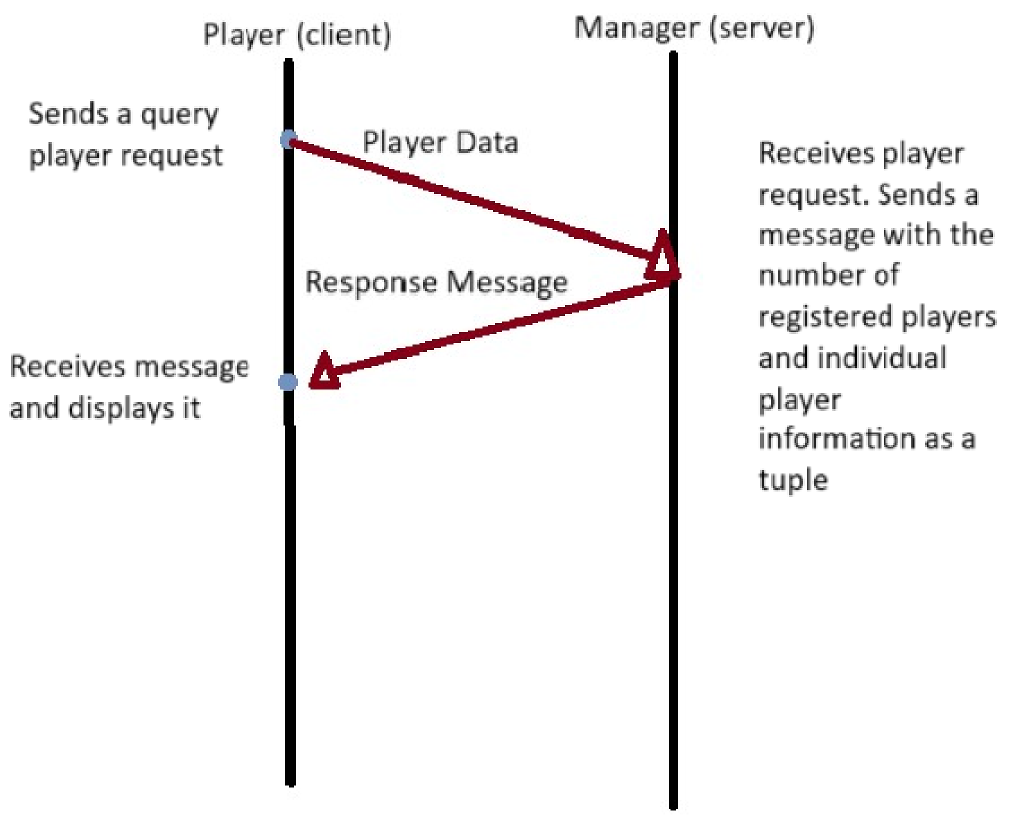
Register command :


Player (client)
Manager (server)
Sends a register player command request with valid arguments
Player Data
Receives player request. Check for duplicates proceeds if no duplicates and sends a response code of success else sends failure
Response Message
Receives response and displays it

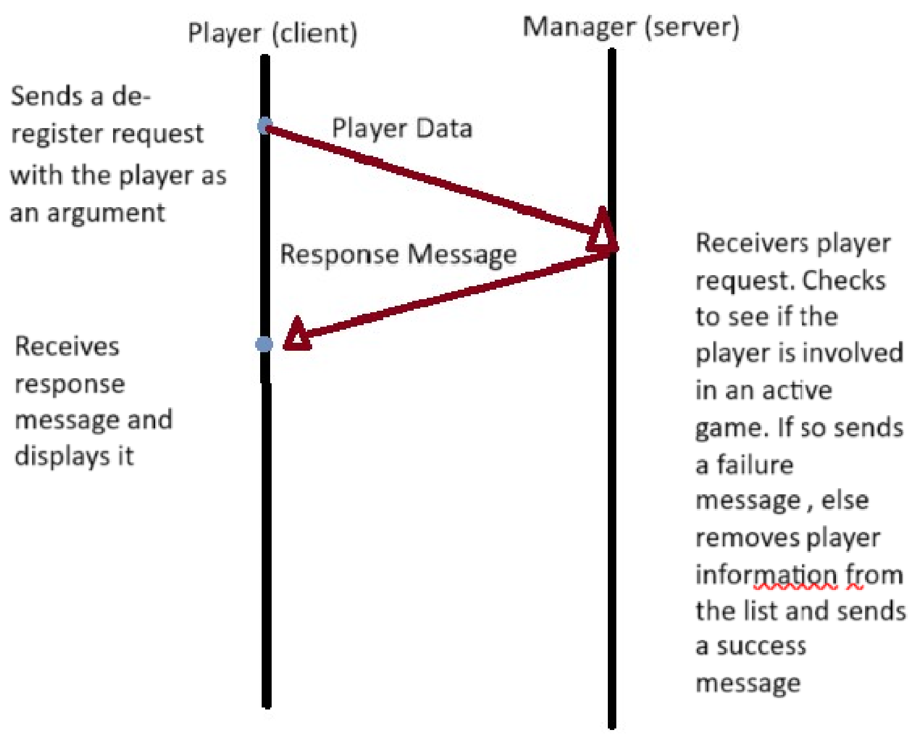
Query Game Command:


Player (client)
Manager (server)
Sends a query game command
Player Data
Receives player request. Sends two messages. First one: number of active games . Second one: an object containing a list of respective game information
Response Message
Receives first message displays it
Receives second message displays it

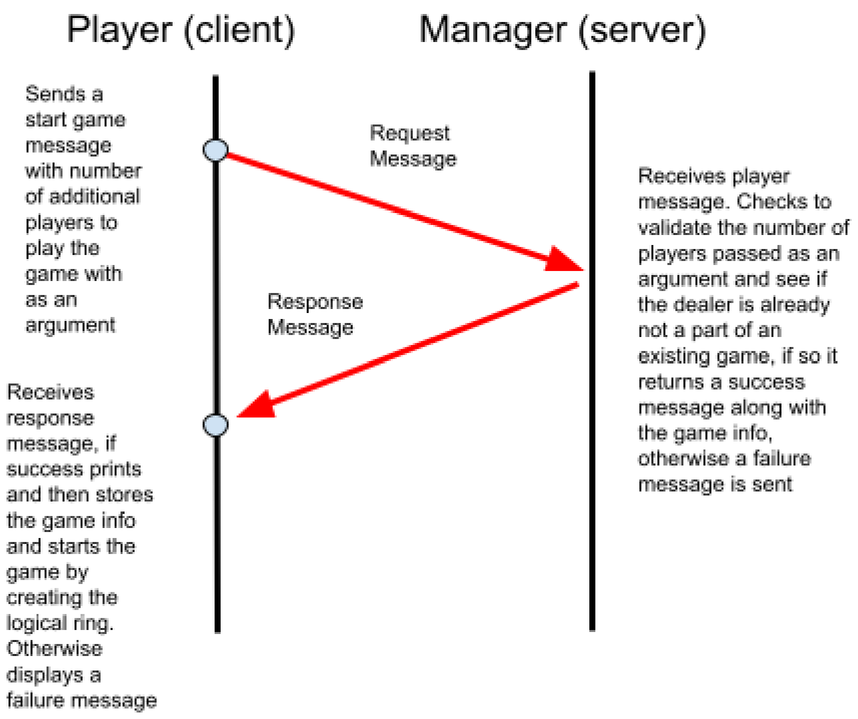
Query Player Command


Player (client)
Manager (server)
Sends a query player request
Player Data
Receives player request. Sends a message with the number of registered players and individual player information as a tuple
Response Message
Receives message and displays it

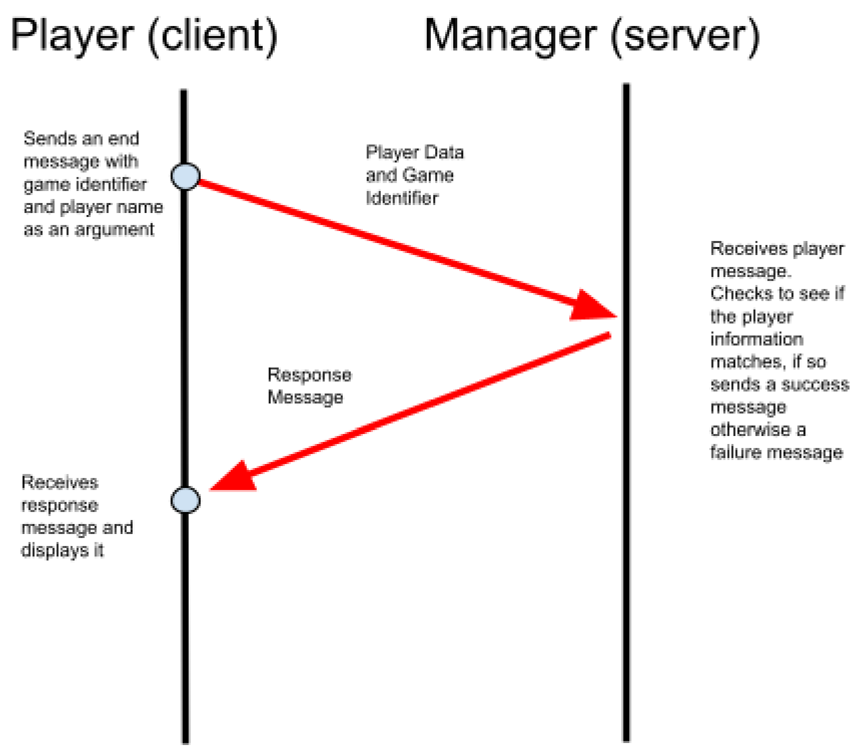
De-register player command:


Player (client)
Manager (server)
Sends a de-register request with the player as an argument
Player Data
Response Message
Receivers player request. Checks to see if the player is involved in an active game. If so sends a failure message , else removes player information from the list and sends a success message
Receives response message and displays it

Start-Game command

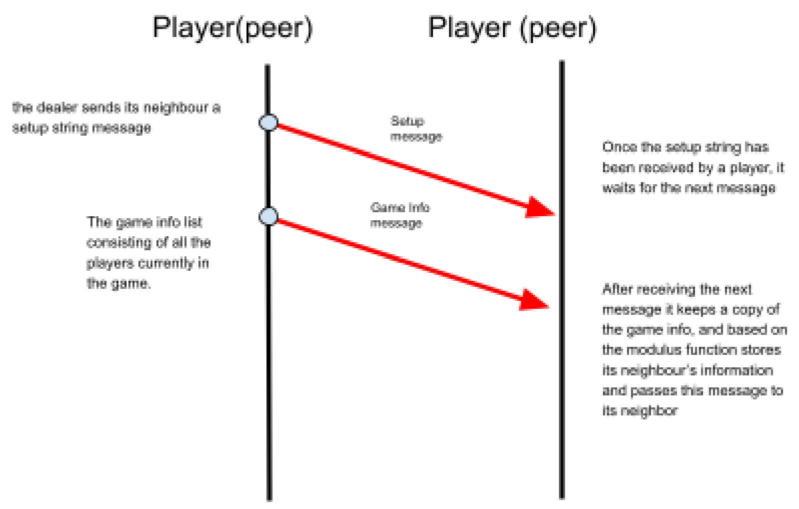

Player (client)
Manager (server)
Sends a start game message with number of additional players to play the game with as an argument
Request Message
Receives player message. Checks to validate the number of players passed as an argument and see if the dealer is already not a part of an existing game, if so it returns a success message along with the game info, otherwise a failure message is sent
Response Message
Receives response message, if success prints and then stores the game info and starts the game by creating the logical ring. Otherwise displays a failure message

End player command

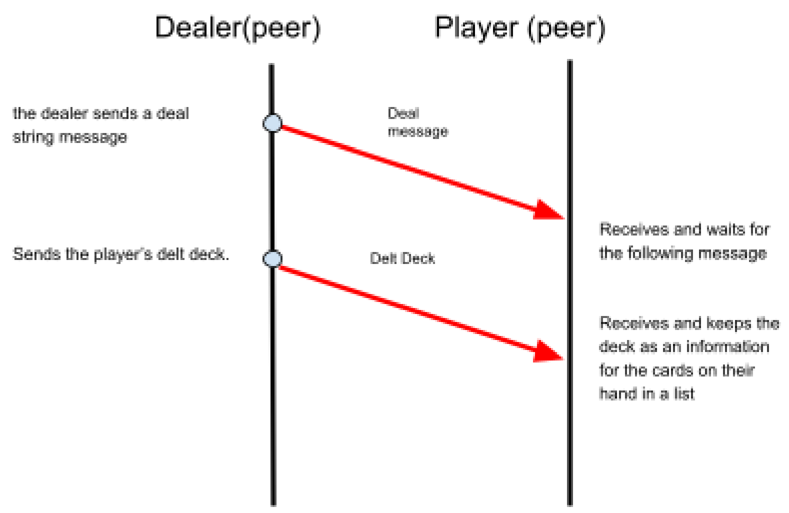

Player (client)
Manager (server)
Sends an end message with game identifier and player name as an argument
Player Data and Game Identifier
Receives player message. Checks to see if the player information matches, if so sends a success message otherwise a failure message
Response Message
Receives response message and displays it

Player Commands

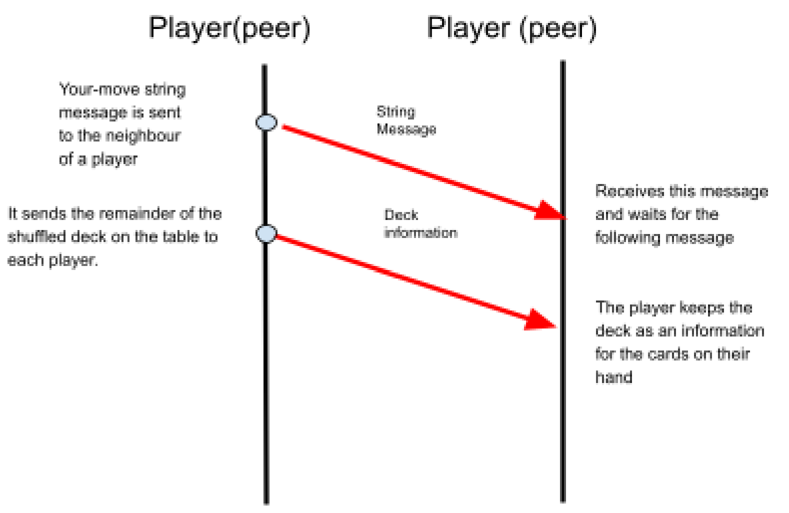
Setup Command


Player(peer)
Player (peer)
the dealer sends its neighbour a setup string message
Setup message
Once the setup string has been received by a player, it waits for the next message
The game info list consisting of all the players currently in the game.
Game info message
After receiving the next message it keeps a copy of the game info, and based on the modulus function stores its neighbour's information and passes this message to its neighbor

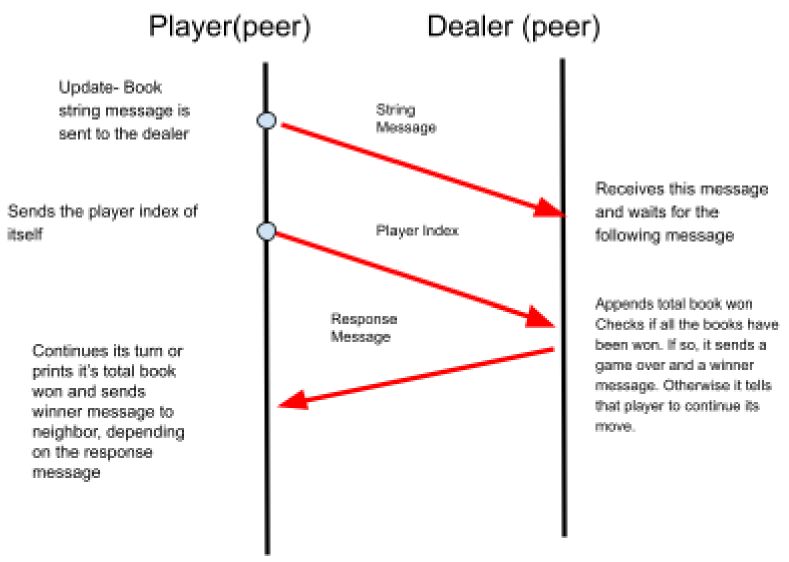
Deal Command


Dealer(peer)
Player (peer)
the dealer sends a deal string message
Deal message
Receives and waits for the following message
Sends the player's delt deck.
Delt Deck
Receives and keeps the deck as an information for the cards on their hand in a list

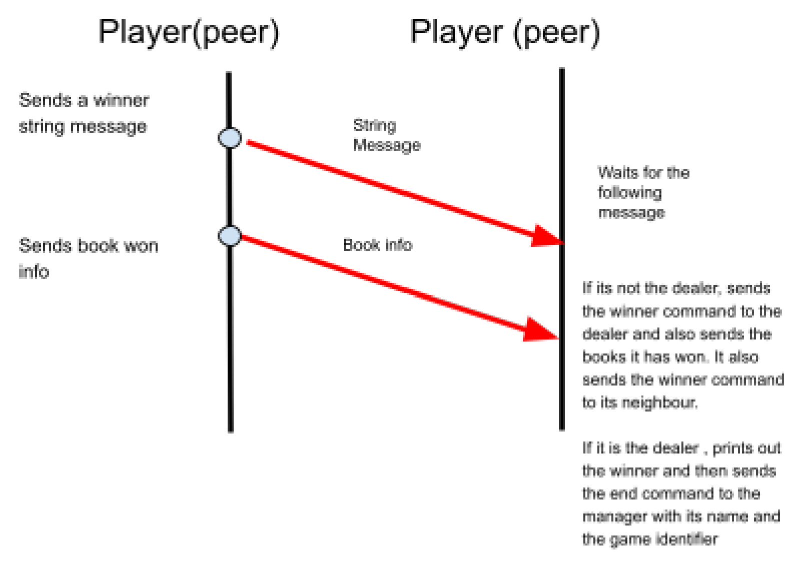
Your- Move Command


Player(peer)
Player (peer)
Your-move string message is sent to the neighbour of a player
String Message
Receives this message and waits for the following message
It sends the remainder of the shuffled deck on the table to each player.
Deck information
The player keeps the deck as an information for the cards on their hand

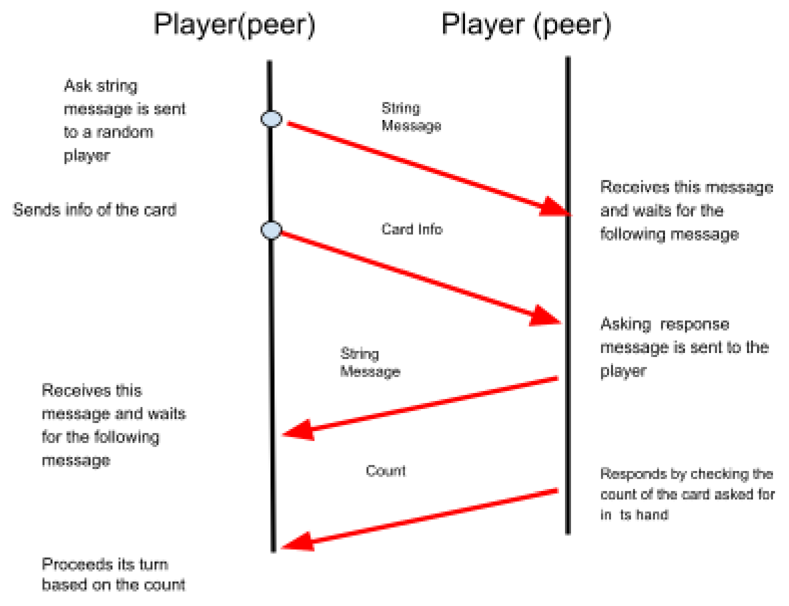
Update_Books command


Player(peer)
Dealer (peer)
Update- Book string message is sent to the dealer
String Message
Receives this message and waits for the following message
Sends the player index of itself
Player Index
Appends total book won Checks if all the books have been won. If so, it sends a game over and a winner message. Otherwise it tells that player to continue its move.
Response Message
Continues its turn or prints it's total book won and sends winner message to neighbor, depending on the response message

Winner Command


Player(peer)
Player (peer)
Sends a winner string message
String Message
Waits for the following message
Sends book won info
Book info
If its not the dealer, sends the winner command to the dealer and also sends the books it has won. It also sends the winner command to its neighbour.
If it is the dealer , prints out the winner and then sends the end command to the manager with its name and the game identifier

Ask Command


Player(peer)
Player (peer)
Ask string message is sent to a random player
String Message
Receives this message and waits for the following message
Sends info of the card
Card Info
Asking response message is sent to the player
String Message
Receives this message and waits for the following message
Count
Responds by checking the count of the card asked for in ts hand
Proceeds its turn based on the count

Design Decisions-

We decided to use lists to store player and game information because they are easy to manipulate and traverse. The message format we chose for communication between client and server was string data using spaces as delimiters to make parsing the message on the manager side as easy as possible. Python can easily turn a string into a list using the split() function. We decided to use the pickle library in python to convert objects into bytes so that we can send list objects in the query games command. We used this library to communicate any list information between peers such as game_info, current rankings, shuffled deck, deck as well cards at hand. All of these objects were in the format of a list. The pickle load was used to convert bytes back to objects and pickle dumb was used to convert objects into bytes. We used game info as a multi dimensional list storing lists containing information about players.

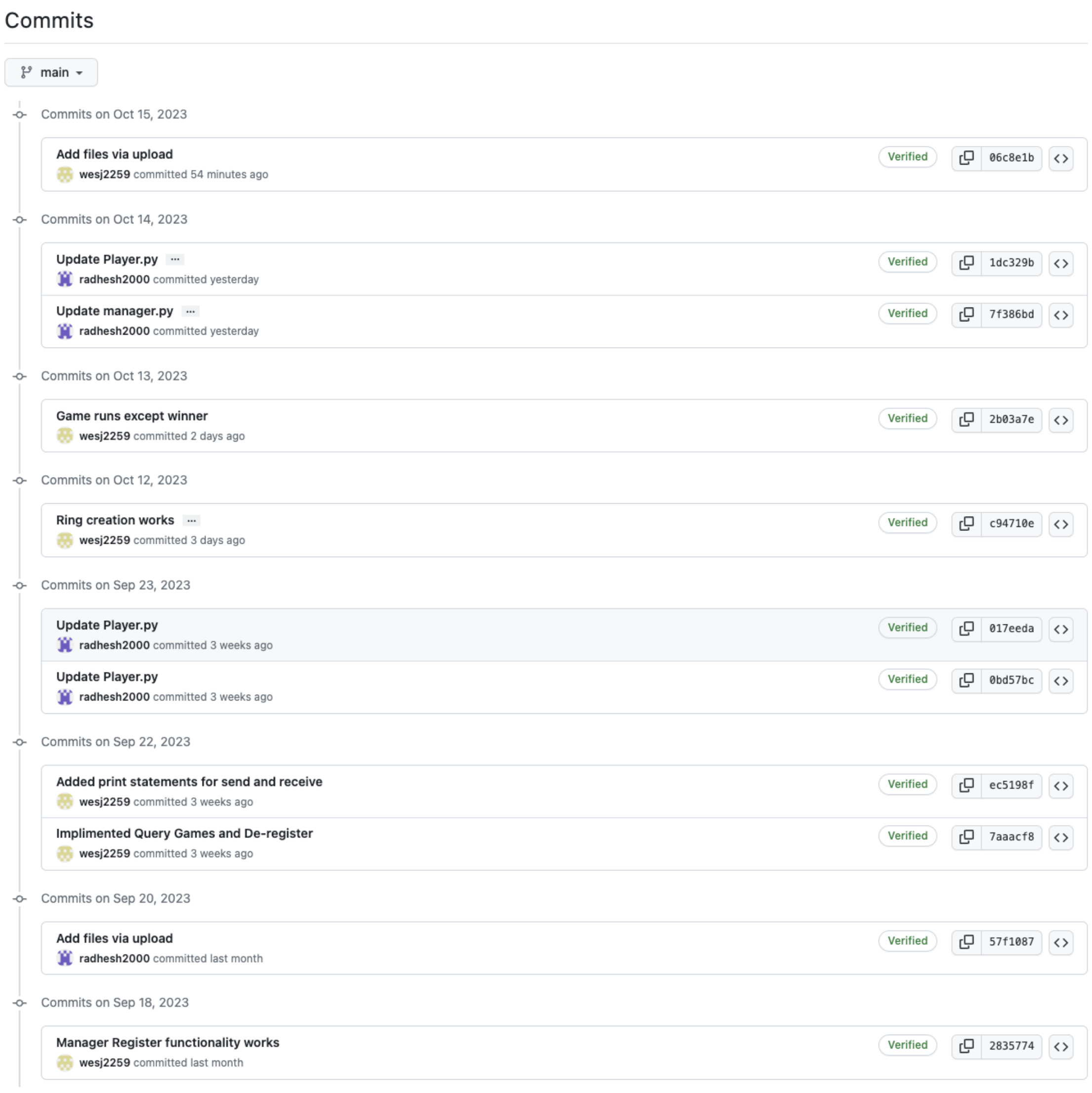
Our implementation is single threaded, to make our code successfully work we initiated our socket to be non blocking, so that the program continues execution even if no commands are being exchanged. In the player menu we have kept a listen option, so that the player peers could listen for any incoming messages. The dealer, when it receives a success message is then able to indicate to the other players listening through the menu about the game info.

For the exchange of information between the players, once it does receive any string command, it then sets the socket to be blocking as the code cannot proceed until it gets the information associated with the command. Once the command is successfully handled it is set back to be non blocking.

We made the game to be computerised. The cards dealt, as well as what card the player is asking for in its turn are all randomised by using the python random library.

After an ongoing game is over each player jumps back to the main menu. This happens by setting the game_in_progress boolean to false which breaks it out of the loop

GitHub Commits-


Commits

main

Commits on Oct 15, 2023

Add files via upload
wesj2259 committed 54 minutes ago — Verified — 06c8e1b

Commits on Oct 14, 2023

Update Player.py
radhesh2000 committed yesterday — Verified — 1dc329b

Update manager.py
radhesh2000 committed yesterday — Verified — 7f386bd

Commits on Oct 13, 2023

Game runs except winner
wesj2259 committed 2 days ago — Verified — 2b03a7e

Commits on Oct 12, 2023

Ring creation works
wesj2259 committed 3 days ago — Verified — c94710e

Commits on Sep 23, 2023

Update Player.py
radhesh2000 committed 3 weeks ago — Verified — 017eeda

Update Player.py
radhesh2000 committed 3 weeks ago — Verified — 0bd57bc

Commits on Sep 22, 2023

Added print statements for send and receive
wesj2259 committed 3 weeks ago — Verified — ec5198f

Implimented Query Games and De-register
wesj2259 committed 3 weeks ago — Verified — 7aaacf8

Commits on Sep 20, 2023

Add files via upload
radhesh2000 committed last month — Verified — 57f1087

Commits on Sep 18, 2023

Manager Register functionality works
wesj2259 committed last month — Verified — 2835774

Video Link:

https://www.youtube.com/watch?v=YbjBgZgFdU4