MULTIPLAYER ONLINE GAME

CSE354: Distributed Computing

Spring 2023

Abstract

This is report discusses how a racing game can implement MMO, massively multiplayer online games, architecture that also makes use of a shared chat feature as well as others.

Sohayla Ihab Hamed	19P7343
AbdulRahman Essam Sayed Heikal	17P6062
Serag Eldin Mohamed Ali Mohamed	19P1183
Yasmin Haitham Abdelmoaty	18P3102

Submitted to:
Prof: Dr. Ayman Bahaa
TA: Eng. Mostafa Ashraf

Introduction

This is a multiplayer online car racing game that uses client server architecture, where each client can connect to the server and the client window shows all other clients' cars and positions on the race track.

An additional feature added is the common chat feature where each client, so long as it's connected to the server, can see what all the chat activity of the other clients. A brief description of the system architecture and coverage is given below.

The system architecture is divided into these parts:

1) Multiclient, multiplayer game states: MainMenu, GameSetup, GamePlay, GameEnd Indices: 0, 1, 2, 3
2) Multiclient chat: Chat features, textboxes and shared data
3) Client-server TCP connection and serialization: All generated sockets are persistent between each client and the server. They are in the ready state with the multiple clients created and prioritized as a queue. Any data that is sent to the socket has to be decoded and serialized.
4) Client-server UDP connection and serialization: Stateless connections are used with each client, so when the client leaves the game or when game is over, the client reconnects and restarts the racetrack.
5) Client database: Records the operation id, the client id, the server id, the score and the date and time it was modified.
6) Latency problems: These problems occur due to many TCP sockets that are created when 100 threads, for example, are concurrently running, each with a maximum of 4 clients.

The system is covered through the following directories:

1) Client: Client state, client index, client GUI..etc
2) Server: Server connection, server protocols..etc
3) Sprites: Images of cars
4) Database: Django files and SQL friendly database
5) Improvements: a draft of improvements that can be later added to the GUI

Table of Contents

List of Figures

Project Description

Outline of Code

The github repository with the code is in this link:
https://github.com/PerfectionistAF/Multiserver-Racing-Game.git

An instructional video:

https://drive.google.com/drive/folders/1UtFu2RX4DXIpXyqCSsIoW-F2_dbnbtkt?usp=sharing

```
main: functional code
1)Client: *GameProxy     *GUI      *main     *Player
           *Protocols     *Proxy
2)Server: *Game           *GameFactory  *GameServer
           *main           *Player       *Protocols      *test
3)Sprites
```

```
database: functional code and database
1)Client
2)Server
3)Database:        *manage            *db.sqlite3
-->3.1)Database: *asgi      *settings
                  *url       *wsgi
-->3.2)RacingGame:*admin *apps
                   *models*tests
                   *urls      *views
          -->3.2.1)migrations:        *001_initial
4)Sprites
```

```
aws: deployment of functional code on aws instance
1)Client
2)Server
3)Sprites
```

Table 1: Outline of code directories and files

Project Beneficiaries

Learning Outcomes

1) Design a distributed computing model to solve a complex problem
2) Design and implement a distributed computing model
3) Configure a working environment for distributed computing
4) Work and communicate effectively in a team

Team Testimony

The project allows us to understand distributed computing concepts and test them out using a fun and interactive game. Along with the distributed concepts, the project enabled us to work on GitHub and to learn and practice using it.

Analysis

Game Features

The game is a top-down 2D car racing game and the average race lasts for 60 seconds. The winner is determined by who raced the most distance in the 60 seconds.

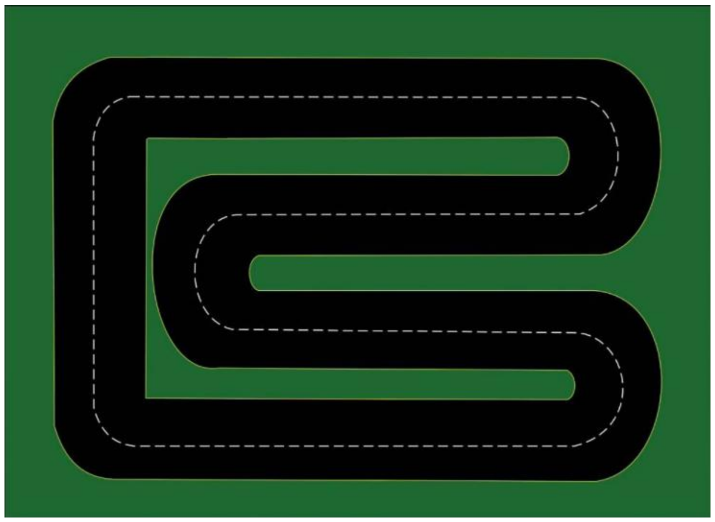


Figure 1: Map of racetrack

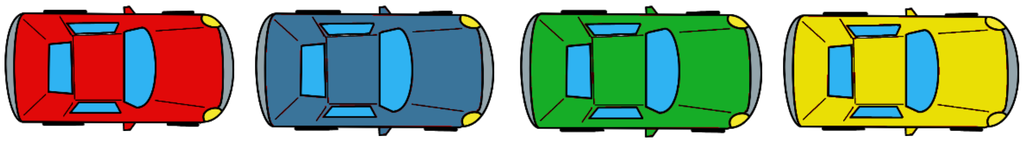


Figure 2: Car sprits in order (0, 1, 2, 3)

Game Storage

The system takes snapshots from the game each interval and stores them in the database. This helps keep track of positions as well as scores of each player.

Game Deployment

The server waits for the TCP request from the clients, if it has the same address that means that the client is already in the game and reconnects them. If the client is new, the server waits until a game is started.

The server has a TCPHandler, and the game setup in client side sends request to the TCPHandler, which goes to either GameFactory or to GameServer through a queuing condition where either queue time (set to 10 seconds) is up, or 4 different players are connected.

Game Functional Code Flowchart

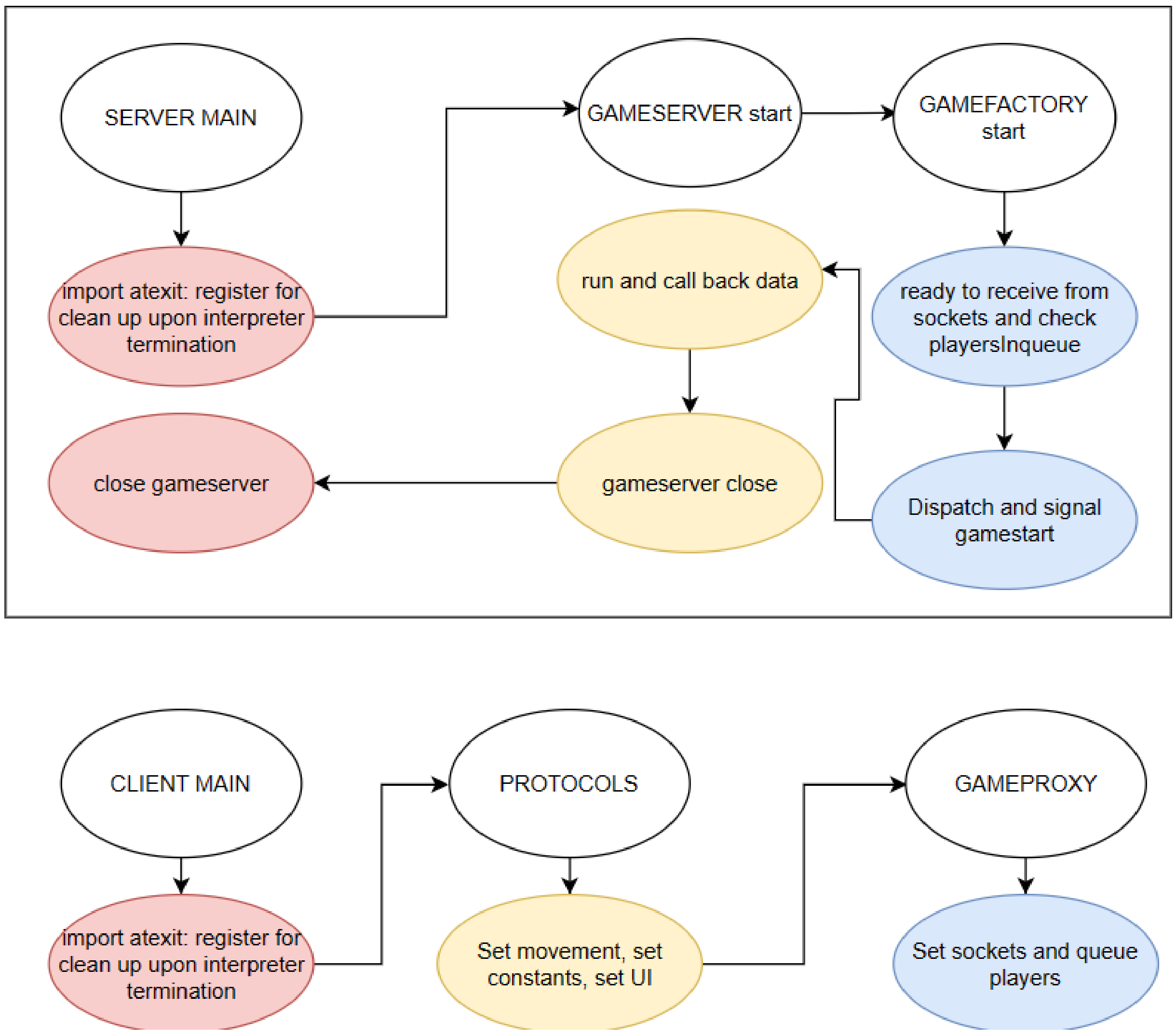

SERVER MAIN
import atexit: register for clean up upon interpreter termination
close gameserver
GAMESERVER start
run and call back data
gameserver close
GAMEFACTORY start
ready to receive from sockets and check playersInqueue
Dispatch and signal gamestart
CLIENT MAIN
import atexit: register for clean up upon interpreter termination
PROTOCOLS
Set movement, set constants, set UI
GAMEPROXY
Set sockets and queue players

Figure 3: Code state diagram

In-Code Directories

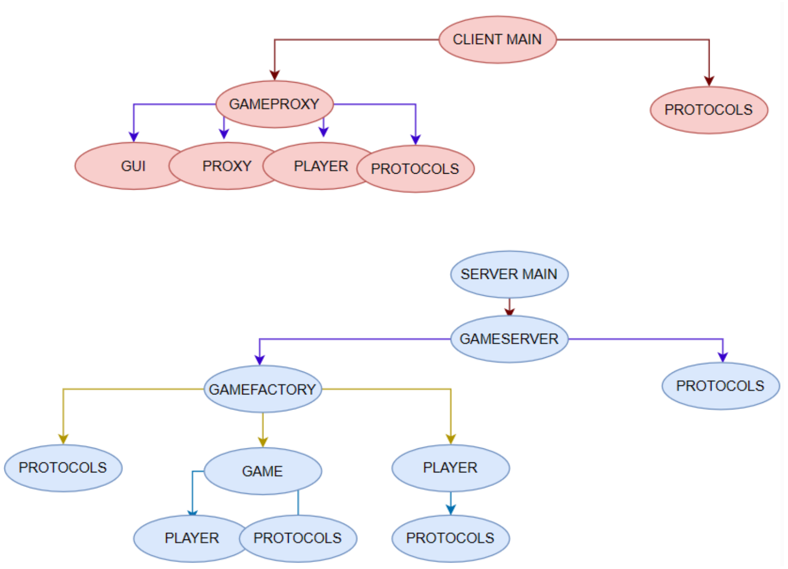


Figure 4: Directory map

Task Plan and Description of Roles

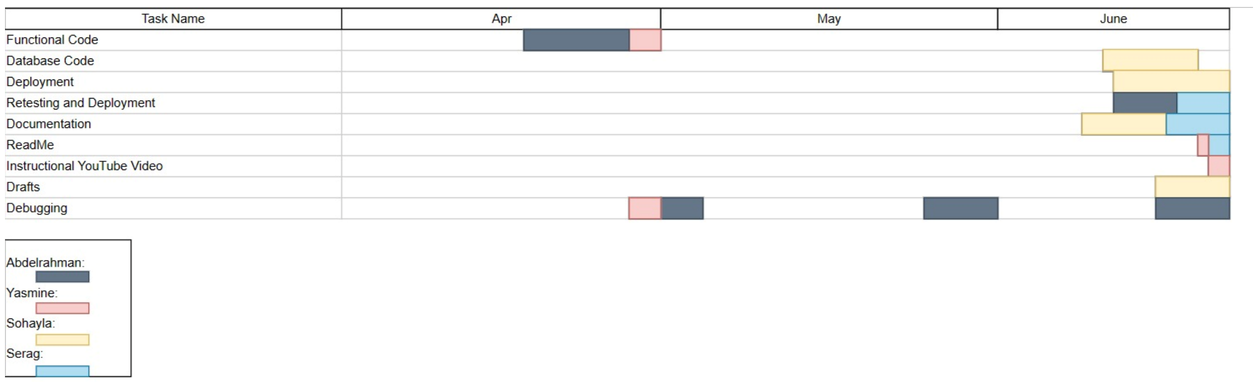


Table 2: Gantt chart describing team member roles and time plan

Game Special Architecture Components

GIL and Thread Switching

The Global Interpreter Lock (GIL) in Python prevents real concurrent execution of threads. Thread switches do not occur between threads in the presence of the GIL. As a result, threads in Python do not run concurrently but rather in a pseudo-parallel manner.

Timeout in GIL

The GIL can cause delays or timeouts in certain scenarios, especially when executing CPU-bound tasks. If a thread holds the GIL for an extended period, other threads may experience delays in acquiring the GIL and executing their tasks.

TCP Socket and Handshake

When a TCP connection is accepted, it results in a socket being created. The handshake process establishes the connection between the server and client.

For example, if the server socket is set to port 8888, each new socket that accepts data through a handshake will have the same port number (8888).This creates a new socket on the server side for communication with each client, and traffic is distributed per socket/client.

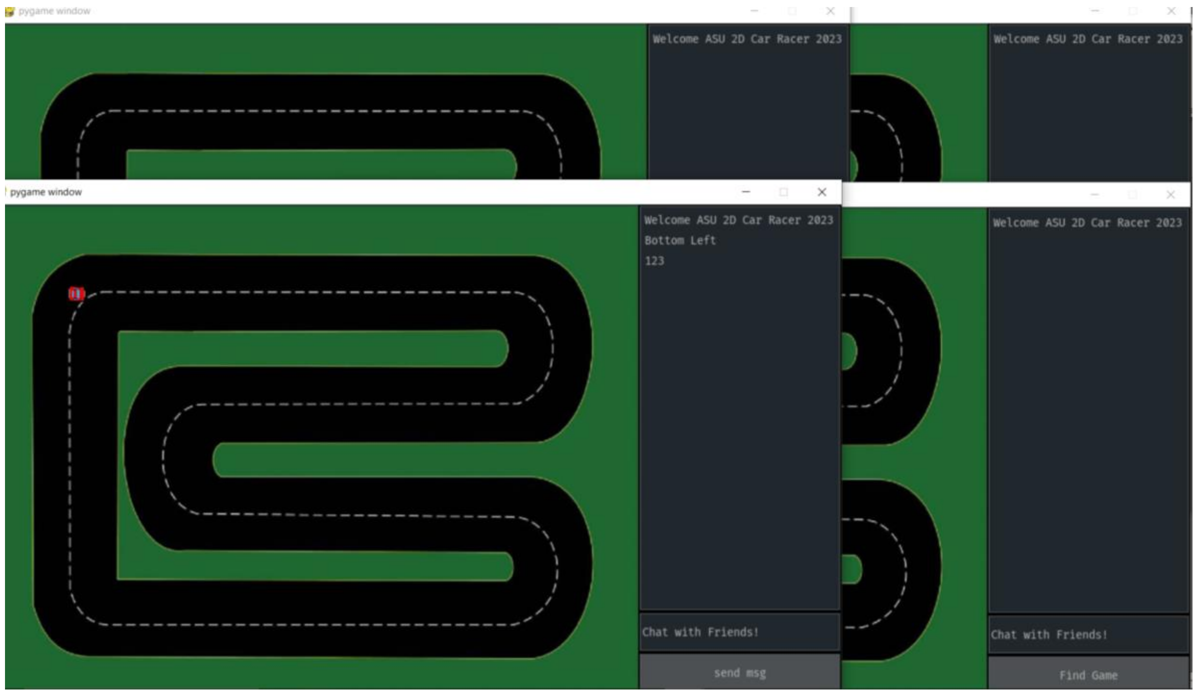
Limiting the Number of Clients

It is important to set a maximum limit on the number of clients that can connect to a server. Allowing an excessive number of clients can consume excessive resources and lead to latency issues. By limiting the number of clients, you can ensure efficient resource utilization and prevent performance problems.

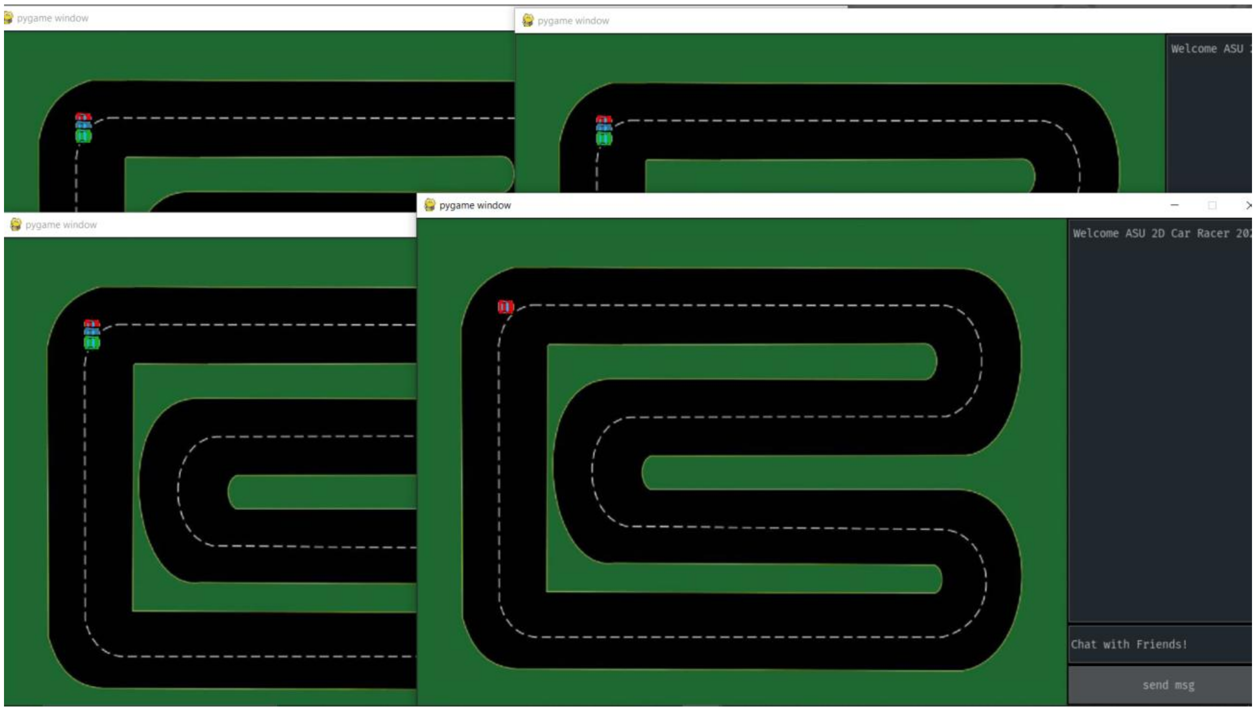
UDP and Stateless Connection

UDP (User Datagram Protocol) is a communication protocol that operates at the transport layer. UDP allows communication with multiple clients without establishing a persistent connection. Unlike TCP, UDP is stateless, meaning it does not maintain session information or perform handshakes. This statelessness can be useful for certain scenarios where persistent connections are not required or when low overhead is desired.

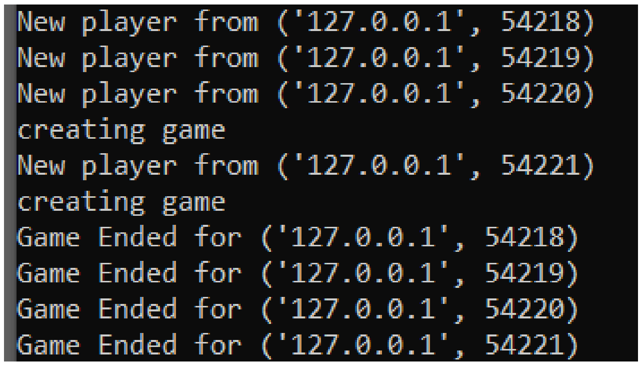
Testing and Deployment


pygame window
Welcome ASU 2D Car Racer 2023
Welcome ASU 2D Car Racer 2023
pygame window
Welcome ASU 2D Car Racer 2023
Bottom Left
123
Welcome ASU 2D Car Racer 2023
Chat with Friends!
send msg
Chat with Friends!
Find Game

Bottom Left alone in game and sending messages, nobody else is in the game so nobody else receives the messages

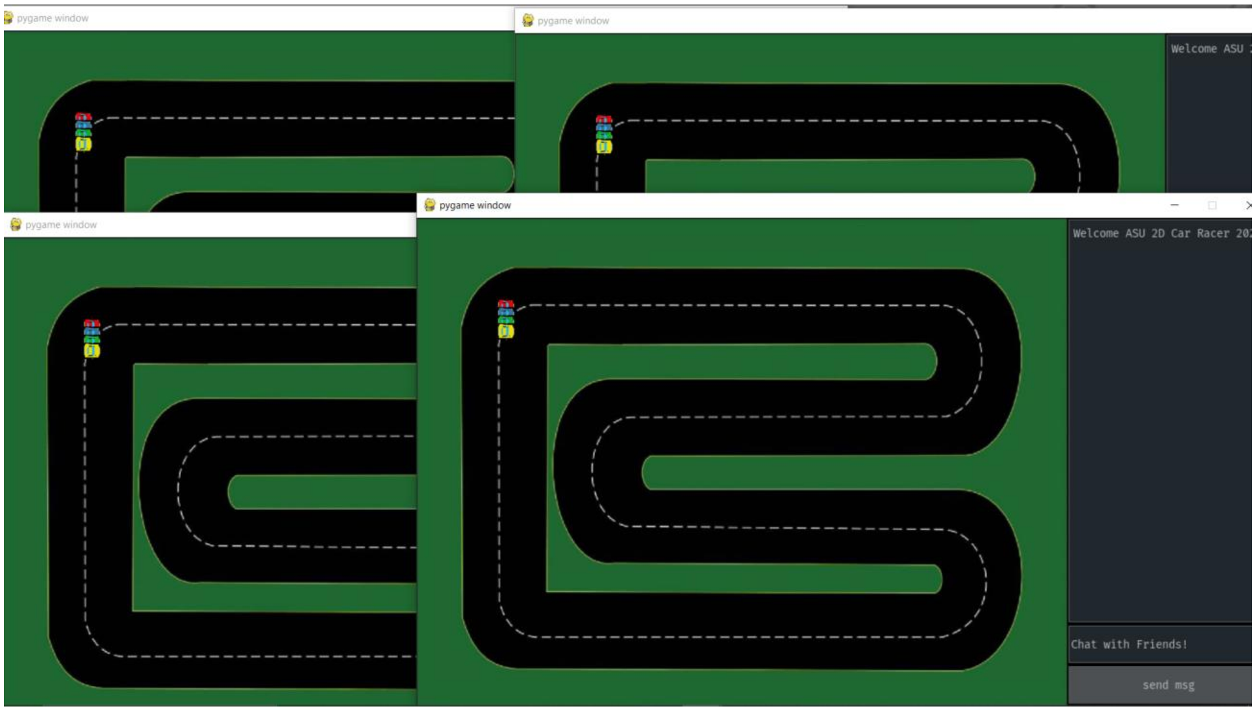


First 3 users clicked on "find game" before the 10 second time limit finished, so they connected together. The last user (bottom right) clicked find game after the 10 seconds, so the server started a different game and after 10 seconds the 4th user entered a game alone.

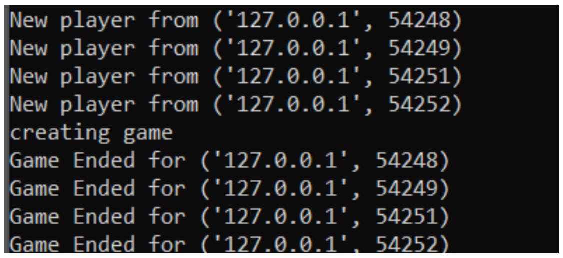


```
New player from ('127.0.0.1', 54218)
New player from ('127.0.0.1', 54219)
New player from ('127.0.0.1', 54220)
creating game
New player from ('127.0.0.1', 54221)
creating game
Game Ended for ('127.0.0.1', 54218)
Game Ended for ('127.0.0.1', 54219)
Game Ended for ('127.0.0.1', 54220)
Game Ended for ('127.0.0.1', 54221)
```

Server side during 2 simultaneous games

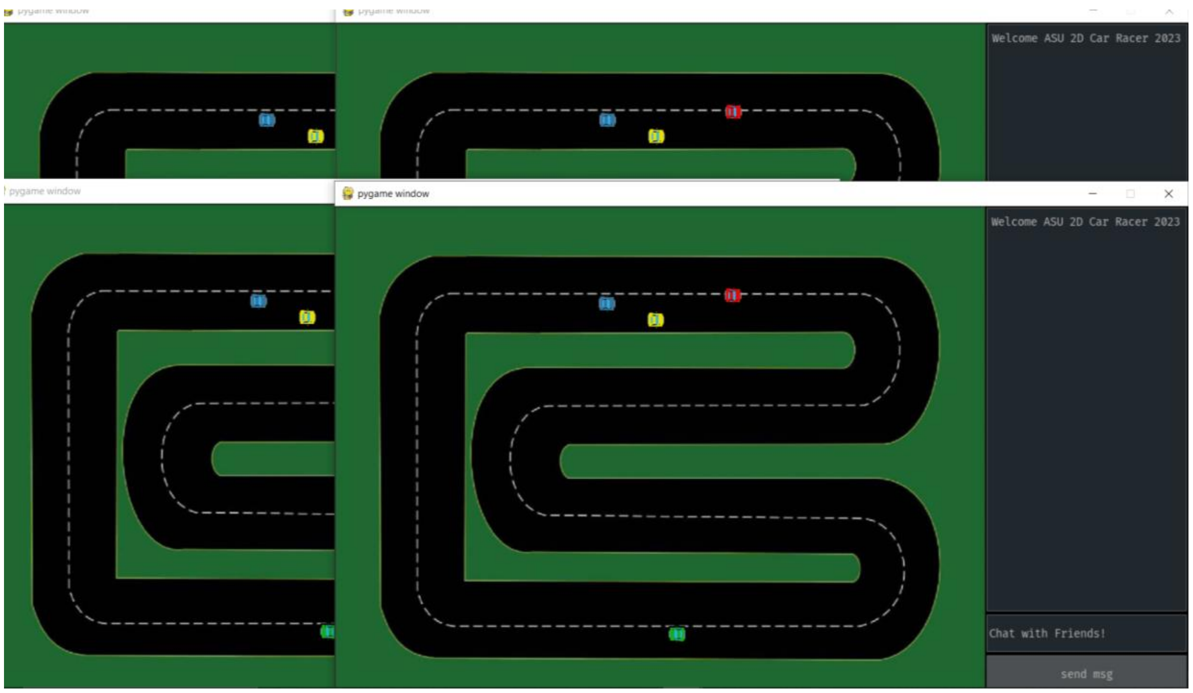


All 4 users started game during 10 second time limit, so the game started immediately after the 4th user connected

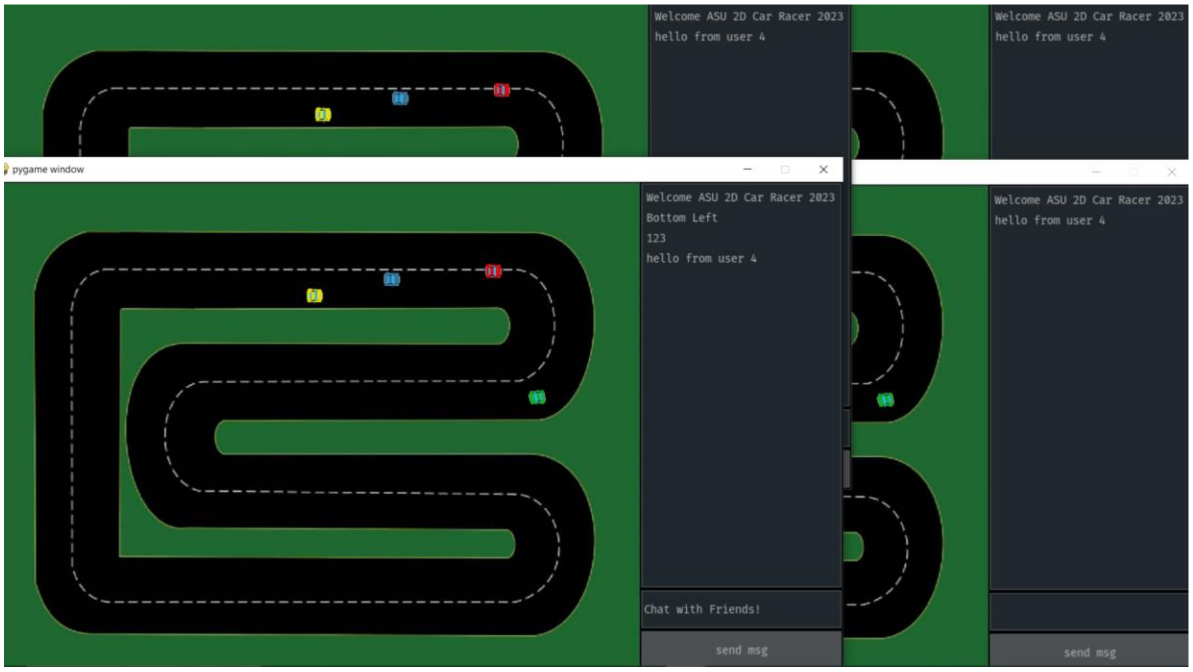


```
New player from ('127.0.0.1', 54248)
New player from ('127.0.0.1', 54249)
New player from ('127.0.0.1', 54251)
New player from ('127.0.0.1', 54252)
creating game
Game Ended for ('127.0.0.1', 54248)
Game Ended for ('127.0.0.1', 54249)
Game Ended for ('127.0.0.1', 54251)
Game Ended for ('127.0.0.1', 54252)
```

Server side during game



green is the winner (most distance raced)

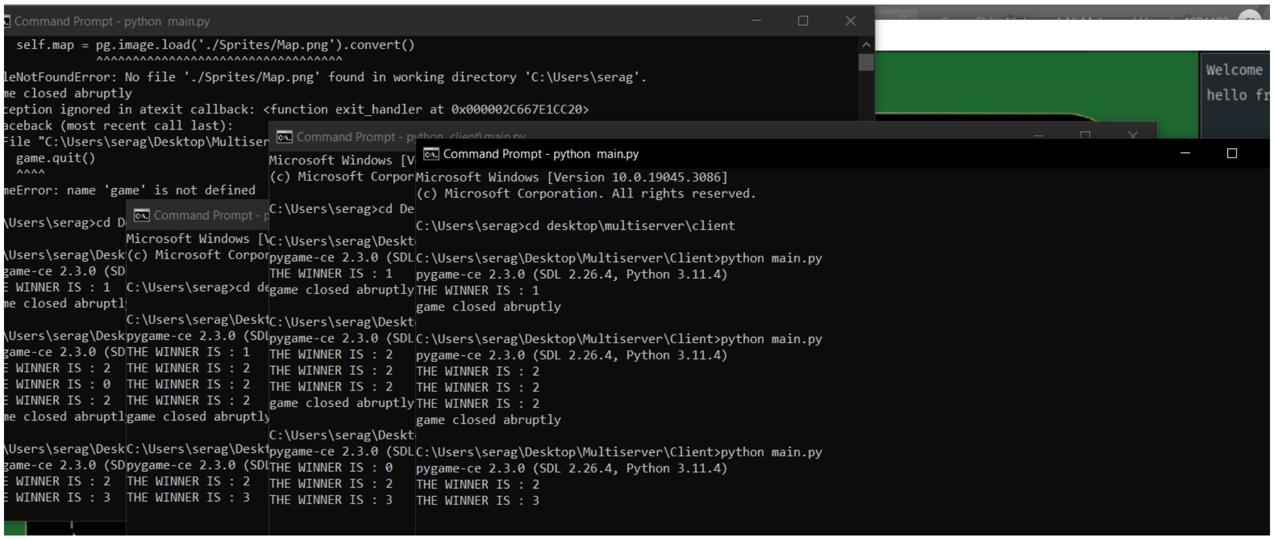

```
Welcome ASU 2D Car Racer 2023
hello from user 4
```

```
Welcome ASU 2D Car Racer 2023
hello from user 4
```

```
Welcome ASU 2D Car Racer 2023
Bottom Left
123
hello from user 4
```

```
Welcome ASU 2D Car Racer 2023
hello from user 4
```

Another race where user 3 won, and user 4 sent message, which was received by all users, since they are all connected



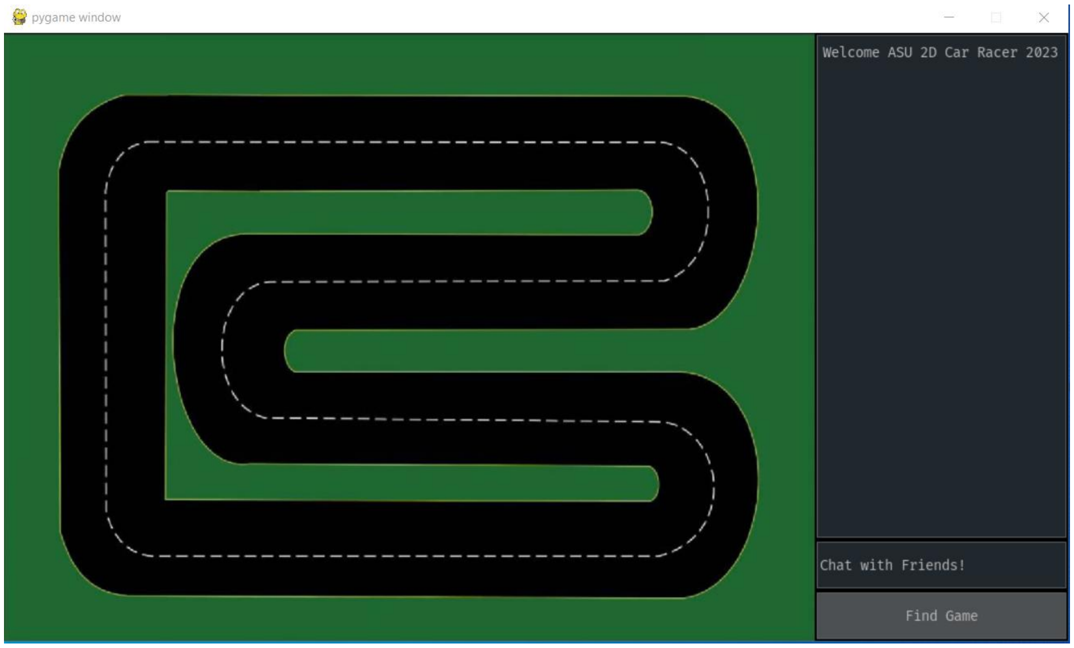
```
Command Prompt - python main.py
 self.map = pg.image.load('./Sprites/Map.png').convert()
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
leNotFoundError: No file './Sprites/Map.png' found in working directory 'C:\Users\serag'.
me closed abruptly
ception ignored in atexit callback: <function exit_handler at 0x000002C667E1CC20>
aceback (most recent call last):
File "C:\Users\serag\Desktop\Multiser
 game.quit()                         Microsoft Windows [V
 ^^^^                                (c) Microsoft Corpor
meError: name 'game' is not defined
                                     C:\Users\serag>cd De
\Users\serag>cd D
          Microsoft Windows [\C:\Users\serag\Deskt
\Users\serag\Desk(c) Microsoft CorporPygame-ce 2.3.0 (SDL
game-ce 2.3.0 (SD                    THE WINNER IS : 1
E WINNER IS : 1   C:\Users\serag>cd degame closed abruptly
me closed abruptl
                  C:\Users\serag\DesktC:\Users\serag\Deskt
\Users\serag\Deskpygame-ce 2.3.0 (SDLpygame-ce 2.3.0 (SDL
game-ce 2.3.0 (SDTHE WINNER IS : 1   THE WINNER IS : 2
E WINNER IS : 2  THE WINNER IS : 2   THE WINNER IS : 2
E WINNER IS : 0  THE WINNER IS : 2   THE WINNER IS : 2
E WINNER IS : 2  THE WINNER IS : 2   game closed abruptly
me closed abruptlgame closed abruptly
                                     C:\Users\serag\Deskt
\Users\serag\DeskC:\Users\serag\Desktpygame-ce 2.3.0 (SDL
game-ce 2.3.0 (SDpygame-ce 2.3.0 (SDLTHE WINNER IS : 0
E WINNER IS : 2  THE WINNER IS : 2   THE WINNER IS : 2
E WINNER IS : 3  THE WINNER IS : 3   THE WINNER IS : 3
```

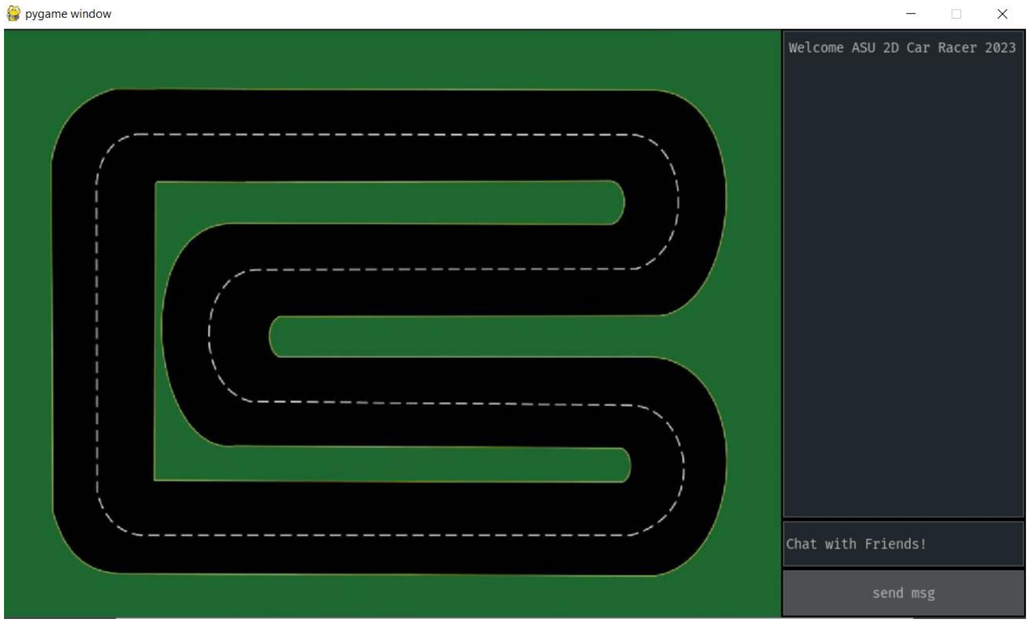
```
Command Prompt - python main.py
Microsoft Windows [Version 10.0.19045.3086]
(c) Microsoft Corporation. All rights reserved.

C:\Users\serag>cd desktop\multiserver\client

C:\Users\serag\Desktop\Multiserver\Client>python main.py
pygame-ce 2.3.0 (SDL 2.26.4, Python 3.11.4)
THE WINNER IS : 1
game closed abruptly

C:\Users\serag\Desktop\Multiserver\Client>python main.py
pygame-ce 2.3.0 (SDL 2.26.4, Python 3.11.4)
THE WINNER IS : 2
THE WINNER IS : 2
THE WINNER IS : 2
game closed abruptly

C:\Users\serag\Desktop\Multiserver\Client>python main.py
pygame-ce 2.3.0 (SDL 2.26.4, Python 3.11.4)
THE WINNER IS : 2
THE WINNER IS : 3
```

All clients display the winner

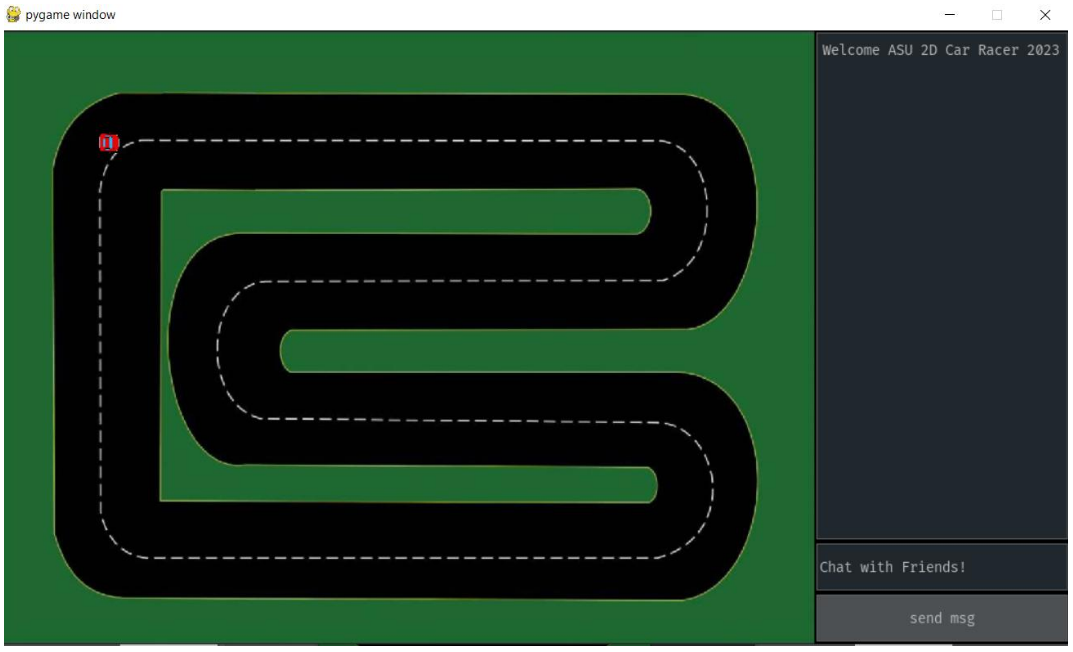
End User Guide

STEP 1: The user then clicks on “Find Game”, which will now become “send msg” after being clicked on.





After 3 other users search for game or 10 seconds pass, whichever comes first, the game will start each player's car will be displayed.



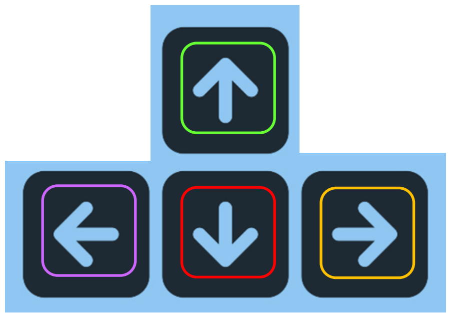
Controls

Up-Arrow: Move forwards

Down-Arrow: Move backwards

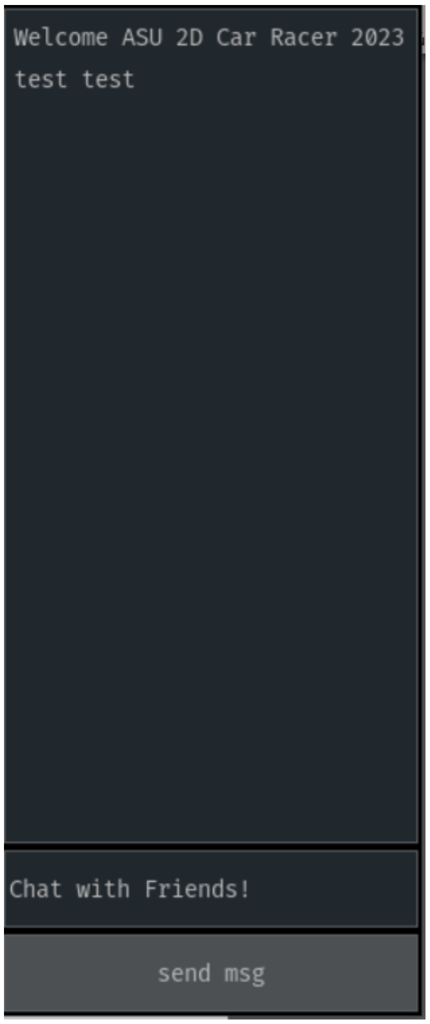
Left-Arrow: Rotate anticlockwise

Right-Arrow: Rotate clockwise



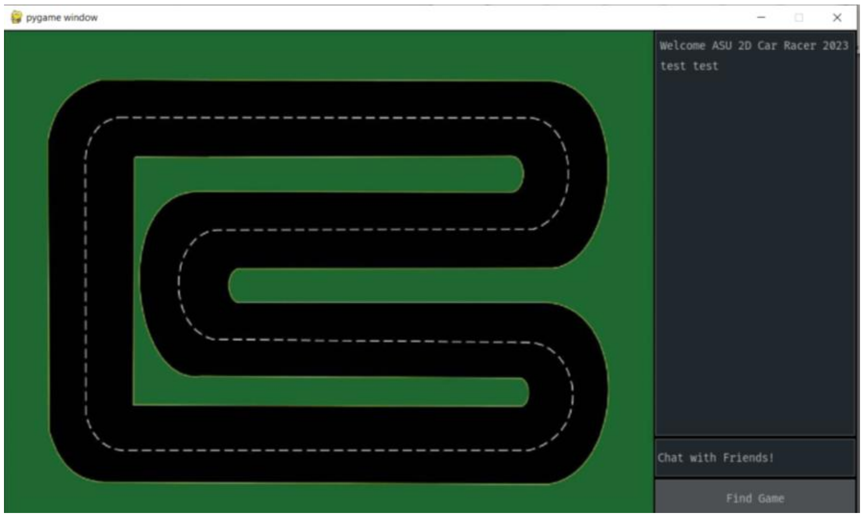
There is also a chat function, only enabled during games, where you can send messages to the other players in the game, and receive whatever they send back.

Here, I typed "test test" in the text box with placeholder text "Chat with Friends!" and clicked "send msg"



```
Welcome ASU 2D Car Racer 2023
test test
```

After 60 seconds, the game ends and the user is returned to the starting screen



Conclusion

In conclusion, our 2D car racing game utilized a client-server architecture, enabling real-time competition among players. This architecture ensured smooth gameplay and seamless communication, allowing players to engage in races. The server efficiently managed game sessions and synchronized player movements, resulting in an engaging racing experience. With our client-server setup, players can enjoy multiplayer racing and showcase their skills on the track. Get ready to experience the adrenaline of our racing game!

References

https://www.youtube.com/watch?v=GRdqOSsjWC4

https://freedns.afraid.org/

https://godotengine.org/asset-library/asset?page=0&category=10&support%5Bofficial%5D=1&sort=updated

https://github.com/tristanbatchler/official-godot-python-mmo/tree/main

https://docs.djangoproject.com/en/4.2/

https://www.pygame.org/docs/

https://pythonprogramming.net/adding-sounds-music-pygame/