REPORT: MAZE CATCH AND RUN

DHANANJAY KUMAR JHA, VATSAL AGARWAL, and YASHASHWEE CHAKRABARTY

CONTENTS

Authors' address: Dhananjay Kumar Jha; Vatsal Agarwal; Yashashwee Chakrabarty.

1 AUTHORS

- **Dhananjay Kumar Jha:** - 2022MCS2059
- **Vatsal Agarwal:** - 2022MCS2056
- **Yashashwee Chakrabarty:** - 2022MCS2057

2 ABSTRACT

One of the most well-liked types of entertainment in the world is multiplayer online gaming. Many players can connect to a game server across the network and make simultaneous moves to compete or collaborate in a shared game environment. Some of the top games include Counter Strike, Minecraft, GTA online etc. We have created a simple multiplayer game(desktop application) using this concept. This is a maze game which currently supports 2 players. As the name suggests, one will be runner and one is catcher ie Danner. The task of the runner is to escape the maze before the Danner could catch him. The maze is built out of boundaries and multiple obstacles that the players can't cross. If the runner is unable to escape that means if the Danner catches him the runner will loose and Danner will win or vice versa. This game supports playing on different machines that is player and Danner can play from different machines or even from the same machine. We have written an autogenerated documentation for all the important modules used in our code using sphinx tool. We have also built some testcases to check the correctness of our application using the unittest library in python. Python language is used for the implementation of this project.

3 SETUP

The setup process is really simple. All external modules in the project are usually shipped with your basic install of python3. Some left is mentioned in the requirements.txt file. The system is only compatible with python3.6 and higher but it should work seamlessly with Linux or Windows.

To begin operation follow the steps below:

(1) Download the zip file and unzip it in the a directory of your preference.

(2) "*cd*" into the project directory using command prompt or terminal and "*cd*" again to go into the *Client* directory.

(3) Run the command *python client.py* from your command prompt or terminal.

(4) Enter your name and the player choice whether you want to be player or Danner.

(5) Repeat these steps for the other player on same machine or other.

(6) Now the game will open and you can easily enjoy the game

General troubleshooting:

- If you are seeing errors like this library not found then just go through the requirements.txt file to check if you have pip install all the needed libraries or not.

4 SOFTWARE ARCHITECTURE and FLOW DIAGRAMS

The Software Architecture and Flow Diagrams have been divided into 3 categories

(1) **Game Architecture**

(2) **Client Server State Diagram**

(3) **Client Flow Chart**

(4) **Server Flow Chart**

4.1   Game Architecture

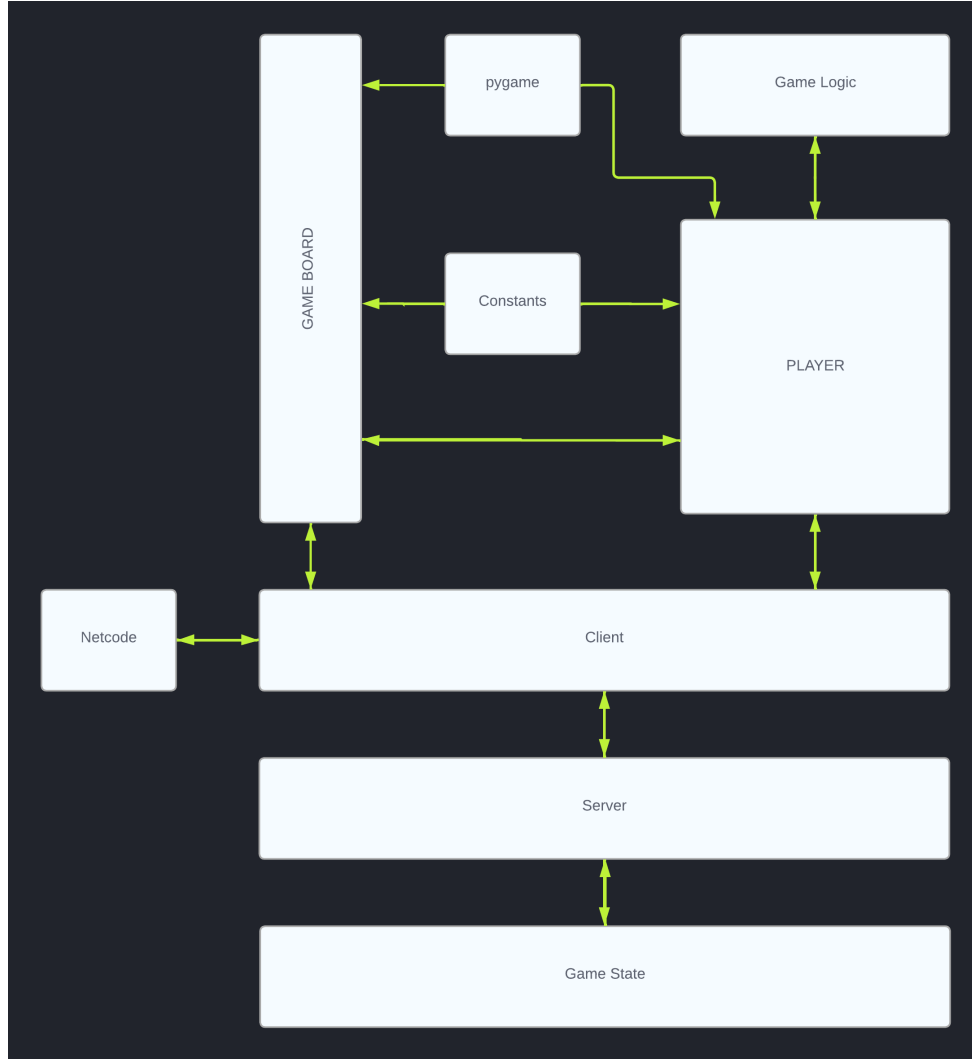Following figure 1 shows the Game architecture of the software:

Fig. 1.   Game Architecture

The Game Architecture has the following components whose description is as follows:

- **Constants**: This component contains the objects of the maze which are going to remain same throughout the game. They mainly include the coordinates of Walls, colors, exit gateway, and hence they are read only objects

which cannot be changed.

- **Pygame**: This component uses the Python Pygame module which is used to make responsive games in python. Further details of Pygame will be mentioned later under External Modules section.

- **Game Logic**: This component consists of the essential game handling logics such as how to move a player using arrow keys, changing speed of the player, setting its position, and various other tasks.

- **PLAYER**: This component is design to incorporate all the features of Game Logic, and constants and with the help of Pygame, it will help set up the Game Board by setting the player on a definite position on the board.

- **Game Board**: This component uses all features of above mentioned components to set up the maze game, which is playable.

- **Client**: This component used to set up a client which will communicate with server to make the game multiplayer on a network.

- **Server**: This component is used to create a server which will communicate with the clients to make the game functional. It also act as an absolute authority between the clients to decide the game state(win,lose,etc.), by checking the collisions between players or player and constants, their updated positions, as send by the clients.

- **Gamestate**: This component is used to decide the state of the game, which is to identify the winner, loser or the updated position. It is handled by the server and is shared to clients, which on the otherside update the state of the players realtime.

- **Netcode**: This component consists of the Prediction and the Rollback strategy which will be used to simulate the game when the players are on high latency networks. Prediction uses a naive approach that for player it will minimize the distance from the exit gate and maximize the distance from the danner, so it chooses the best coordinates out of 8 possible coordinates around it.
For danner, it chooses the coordinate that minimizes the distance between the danner and player.

Rollback is used to check whether predicted coordinate is equal to the recieved coordinate, and updates the position accordingly.

4.2 Client Server State Diagram

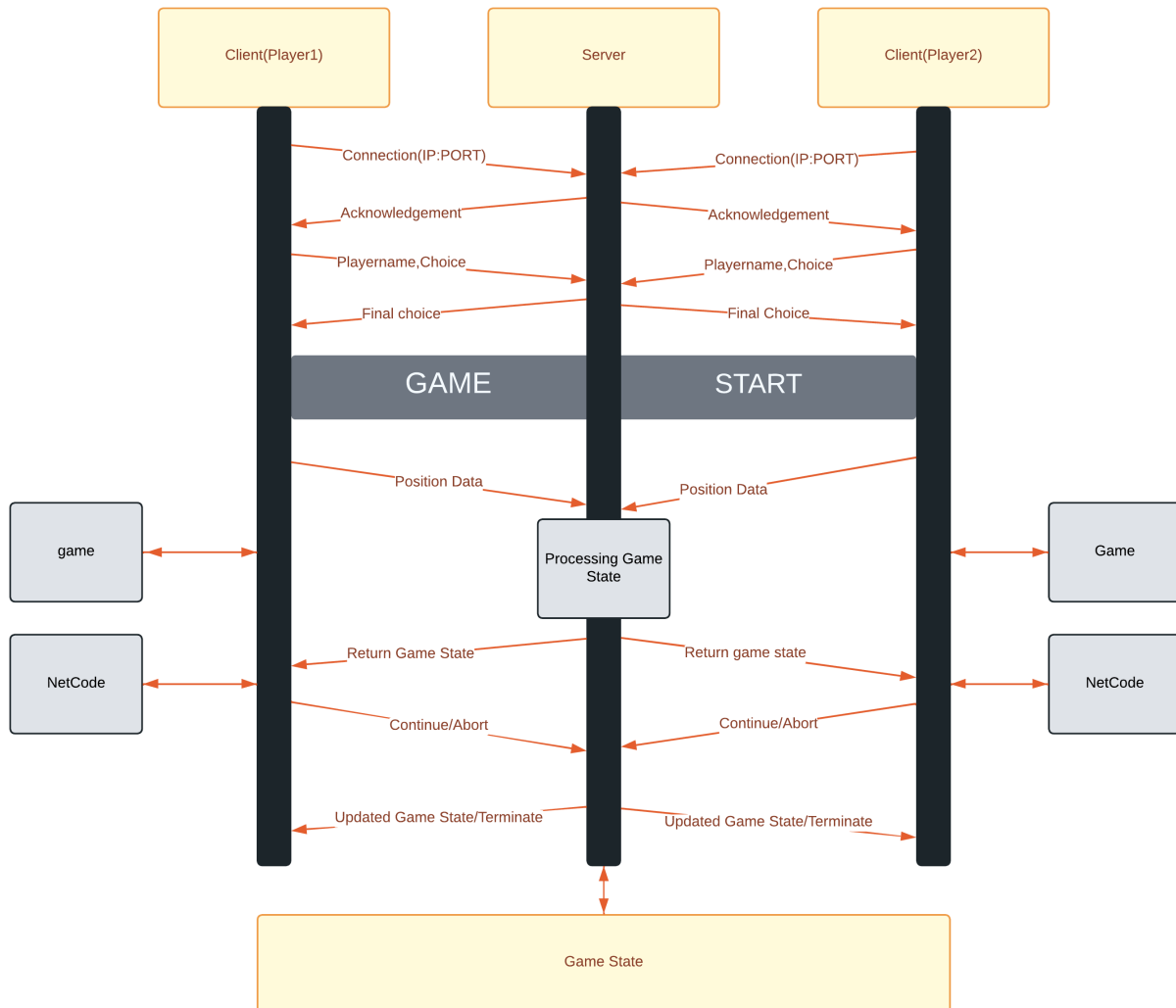Following figure 2 shows the Client Server Diagram of the software:

Fig. 2. Client Server State Diagram

- The client-Server State Diagram states, how the clients and server are communicating on the network, which
  will help to simulate the multiplayer game on different computers.

- Initially both the clients send their connection request, which is further acknowledged by the server using TCP 3-way handshaking method.

- After getting acknowledgement, Players send their User name and choices whether they will be player or catcher, and server returns the final choice to both the players.

- After recieving the final choice, game is started and both the clients send their positional data of the player and catcher respectively.

- With the positional data, server processes the game state, whether there is a win, loose, or game continues, and broadcast the processed state to the clients.

- Clients on the other hand update the game board, which is viewed by the users, so that they view the results of the game at realtime.

- If there is any delay in the packets, Client maintains a frame counter which it uses to predict the other player moves and updates the original position using rollback strategy, when it receives the packets.

- Any error in the connection leads to the abortion of the game and players have to restart the game.

4.3   Client Flow Chart

Following figure 3 shows the Client Flow Chart of the software:
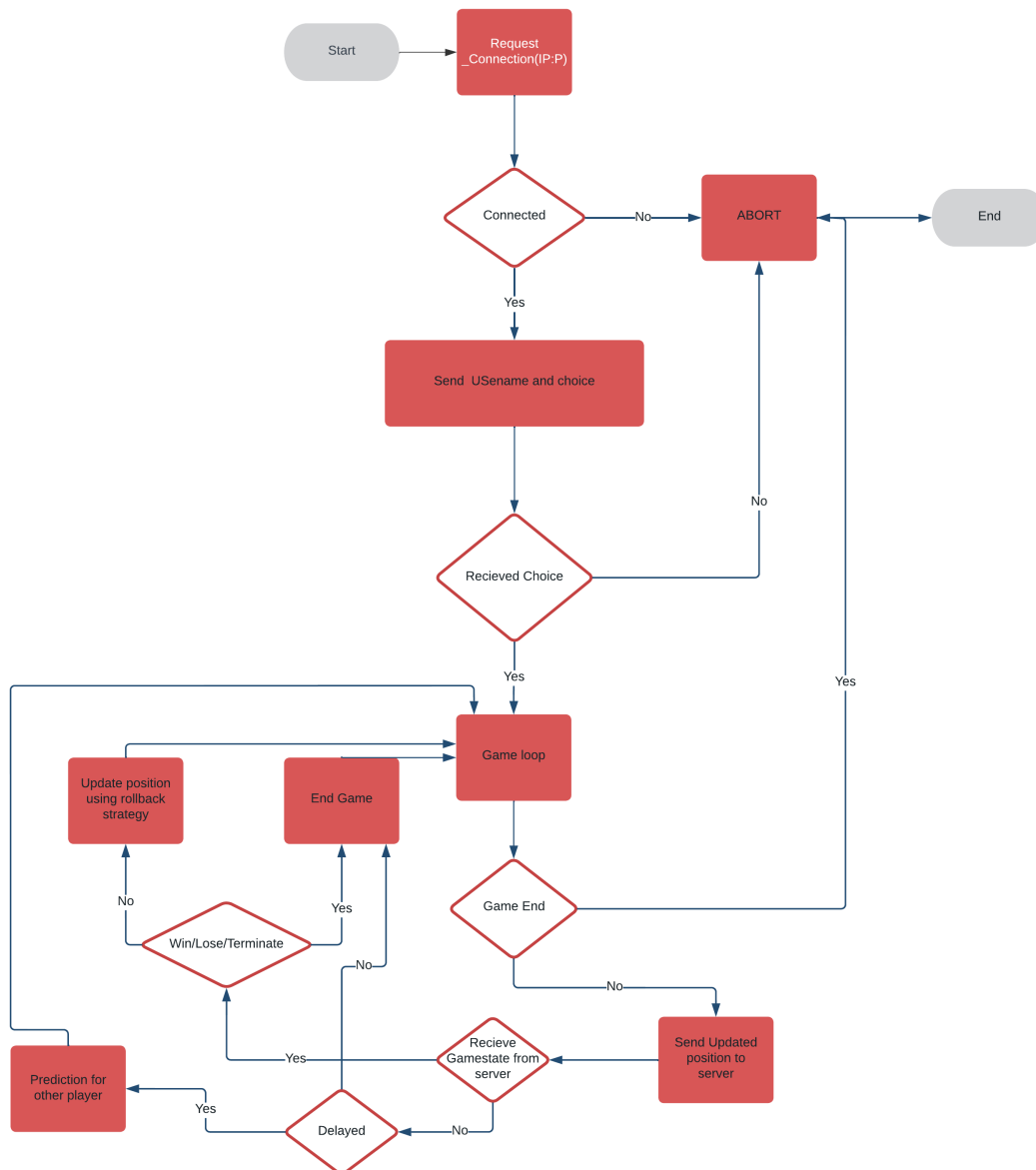

Fig. 3. Client Flow Chart

The client is an autonomous unit, that communicates with the server the make the game responsive, among the two players on different computers.

- Connection: Initially Clients sends the connection request to communicate with the server, if acknowledged then it proceeds further otherwise abort.
- Sending initial Information: After getting connected, clients sends the Username and its choice to the server, to which server acknowledges by sending the final choice.
- Gameloop: At this stage Game is started, and it runs until its a conclusion or any connection errors which is encountered.
- Send Positions: At this stage, client sends the postional data of the players which are updated by the users. To which server replies by sending the updated game state.

- If the updated game state is delayed, then client uses the prediction logic to update the other player positions, so as to make the game tunning smoothly.
- If the updated game state recieved is a game conclusion(i.e win or lose or terminated), then client ends the game by displaying appropriate message to the users.

- If the game is not concluded , then the client updates the player positions using rollback strategy(if the player was on high latency network), and returns back to the game loop, which continues till the game end.

4.4   Server Flow Chart

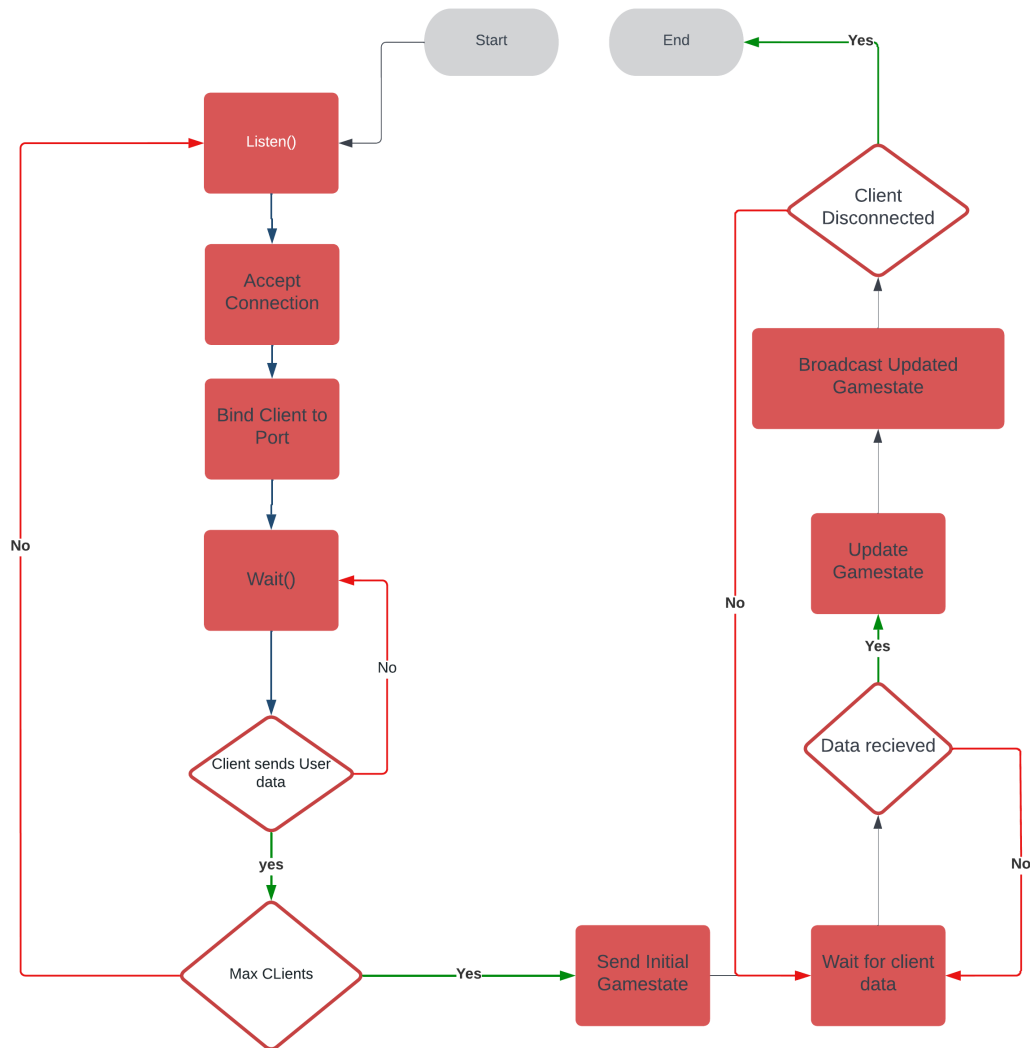Following figure 3 shows the Server Flow Chart of the software:

Fig. 4. Server Flow Chart

Server Flow of Control: The server is an autonomous unit that has the supreme power when it comes to making decisions about the game-state. The server has the following distinct operations:

- Listen: The server waits for a client connection request on a designated address and port.
- Bind: The server binds every client to a certain empty port, from here on every communication from this client will happen through their designated ports.

- Wait and User data: The server waits for the client to send data containing user info like name and their choice (danner or player).

- Send initial state: Finally when the max number of players allowed (2 in our case) has been reached the server broadcasts the initial game states to every client.

- Wait for next Data: The server wait for the next key data from either of the clients. Once received the server updates the game state on its side and broadcasts the state. Note that it broadcasts data every time the game state is changed irrespective of who changes it

- Client Disconnect and abort: If at any point one of the clients disconnect the game is no longer playable and must be aborted. On diconnect the server informs the other client that his friends has disconnected and aborts.

4.5   Rationale

Building the project in the above design gives us the following advantages

- Flexibility to make changes: The server and client code bases are almost completely separate, with minimal code sharing. Hence different team members can edit either without breaking the others code. The only restriction being the rules for the data that is being shared.

- Easier to debug: Debugging becomes easy due to the simplicity of the design. Every module shares bare minimum dependencies. Once data is handed over from one module to the other, we can easily pinpoint where the code breaks.

- Distribution of Work: The Above design was created while keeping the different strengths and weaknesses of the team members in mind. The module responsibilities were distributed accordingly.

5 EXTERNAL MODULES USED

The following section describes the external modules that were used while developing this project.

(1) **SPHINX** This lets us create insightful and meaningful documentation. It can be used with many of the languages (most popularly used language is python). Using some themes, search bar functionality can also be implemented so that we can search anything from the documentation for example classes, functions etc. Some of its major features are:

- *Hierarchical Structure*
- *Themes*
- *Extensions*

(2) **UNNITTEST** It lets us write extensive test cases for our application. Using this, we can break our application into various little modules and test each module separately considering each of them to be a unit. Some of its major features are:

- *Automated*
- *Repeatable*
- *Easy to implement*

(3) **OS,SYS,Platform** The OS library lets us manipulate the file system to create, access, and delete files which is necessary to maintain our markdown repository. The system library is used for giving command line arguments to our programs so that there is a way for the modules to communicate between themselves. Finally, Platform library is used to make our code platform independent. There are minute changes required in the code to make it run on linux or windows, we use the platform library to check which os is running the code.

(4) **PYGAME** This lets us create games using python.It is a list of cross platform python modules that means it can be used on different machines with different packages. It provides sound and graphics libraries created specifically for use with python language. Some of its major features are:

- *Easy and simple to implement*
- *Portable*
- *less amount of code*

(5) **FLASK** makes it simple to create web applications. Basically it provides various tools , libraries , technologies etc which are necessary for building web applications. Some of its major features are:

- *Scalable*
- *Flexible*

(6) **GUNICORN** It lets us run our python desktop application parallelly by executing many python processes in a single dyno. Some of its advantages are:

  - *Less CPU intensive*
  - *Easy to configure*

(7) **Flask-SocketIO** It lets our application support bidirectional low latency communications between the server and the client. Some of its features are:

  - *Reliable*
  - *Supports broadcasting to a group of clients or even all the clients*

(8) **EVENTLET** It is a networking library with the help of which we can change how we execute our code but now how we write it.
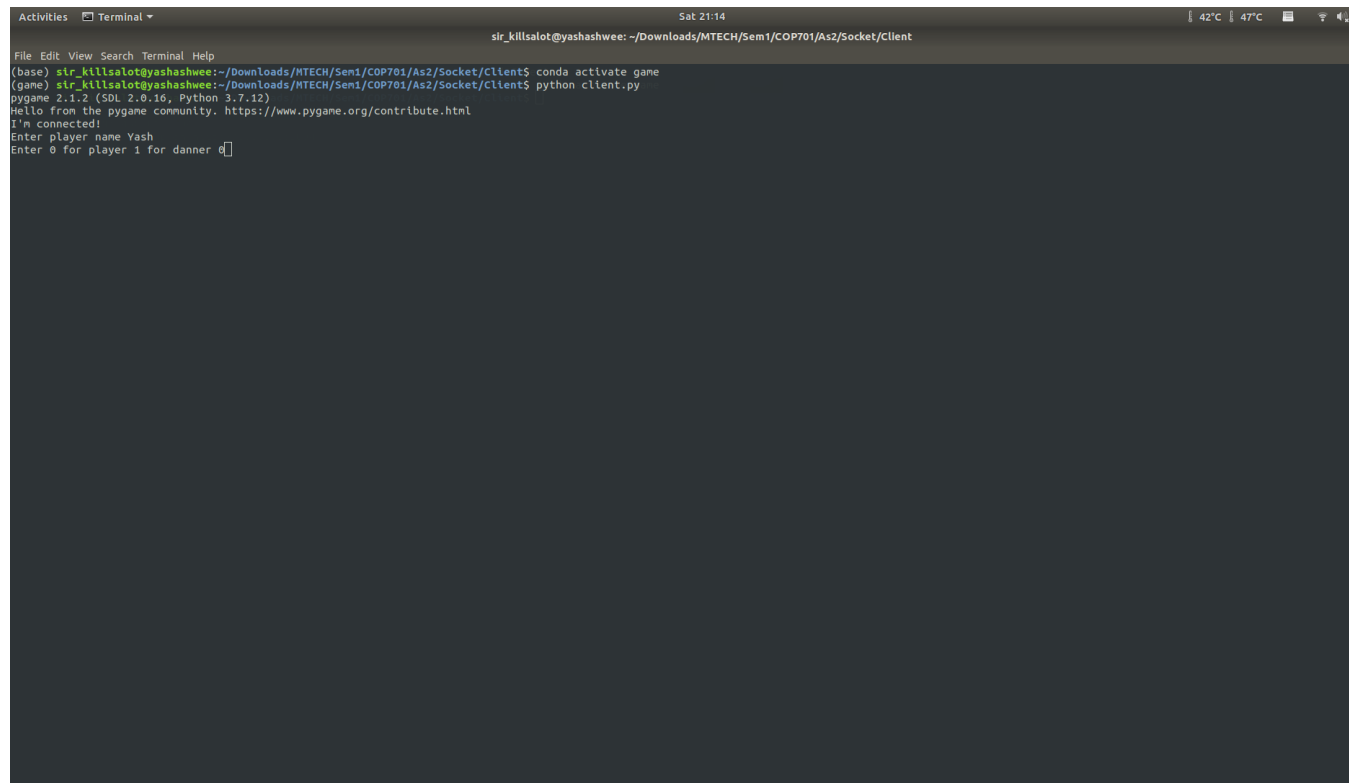
(9) **GEVENT-WEBSOCKET**

6 INTERNAL MODULES USED

Following section describes the internal modules built by the team for this specific task.

(1) **Client** It contains all the code concerning the client side of the game application. This includes all the objects used in the game, their state changes, all the movement functionality, color combinations used etc.

- *client* Conducts communication with the server and updates the private gameboard accordingly
- *players* Contains methods and specifications for setting up the two players, update their positions and check if their move is legal or not.
- *room* Contains the methods and specifications for setting up the game screen with all the obstructions and boundaries
- *rgb* Contains display definitions for colours used in the game

(2) **Server** It contains all the code concerning the server side of the game application. This includes how the server will handle the various things we did in our code like the player movement, error messages etc.

- *flaskserver* The main brain of the server that contains events through which clients talk to the server and update their states. It also contains logic to test gamestates, and has the final say when it comes to player position.

(3) **app** This is a replication of the flask server built to run on a heroku dyno on the internet.

(4) **tests** This contains all the extensive tests written by us on our application. It includes:

- *connection*: Tests server client connections
- *game conclusion*: Tests game conclusion logic (Player wins)
- *collision test*: Tests collision logic (Danner wins)
- *server Events* : Tests all the events on the server and compares their outputs using a set of predefined inputs

(5) **Docs** This contains all the things required for auto-documentation. It contains documentation about the classes, functions and their parameters, tests etc. We have divided the documentation in three parts:

- *Introduction*
- *contents*
- *tests*

7 DEMO

7.1 Starting Game

It will ask for the player name and his/her choice whether he/she wants to become player or Danner. 0 for player and 1 for Danner.

Fig. 5. Input Screen

7.2 Game Window

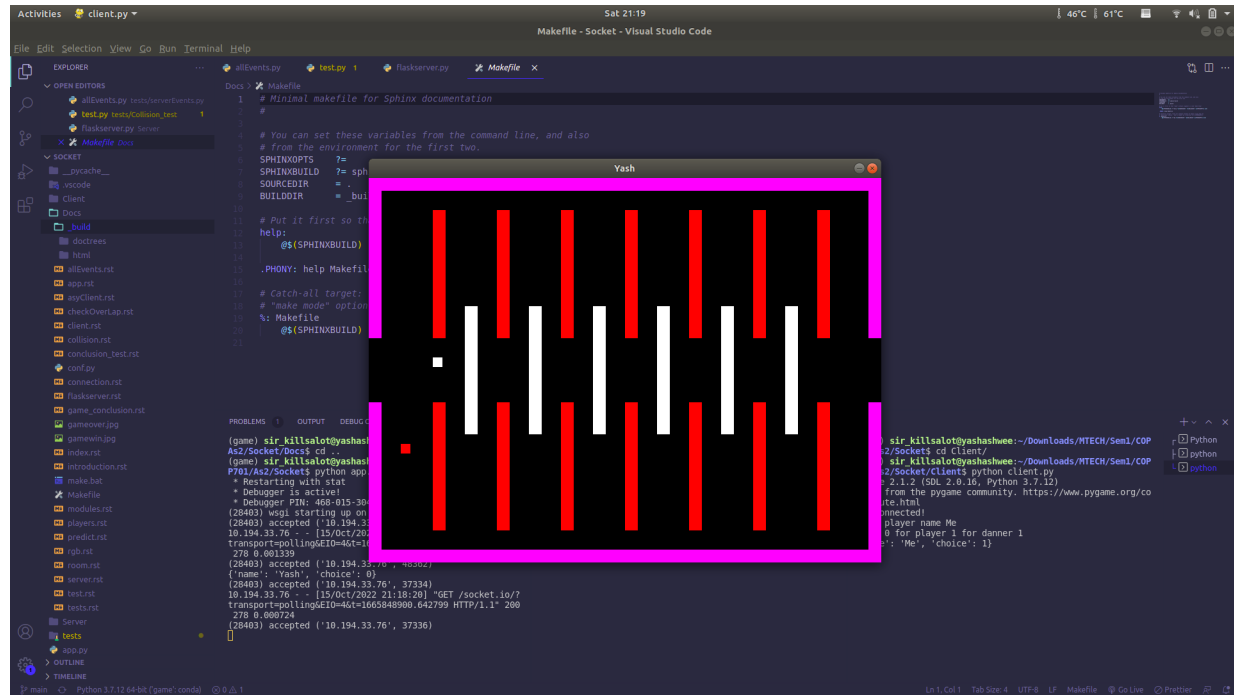After entering the details, game will start and the game window will be displayed.

Fig. 6. Start Window

7.3 Collision - danner wins

If Danner catches player the collision will happen and on danner screen win screen will show and on player screen lose screen will show.
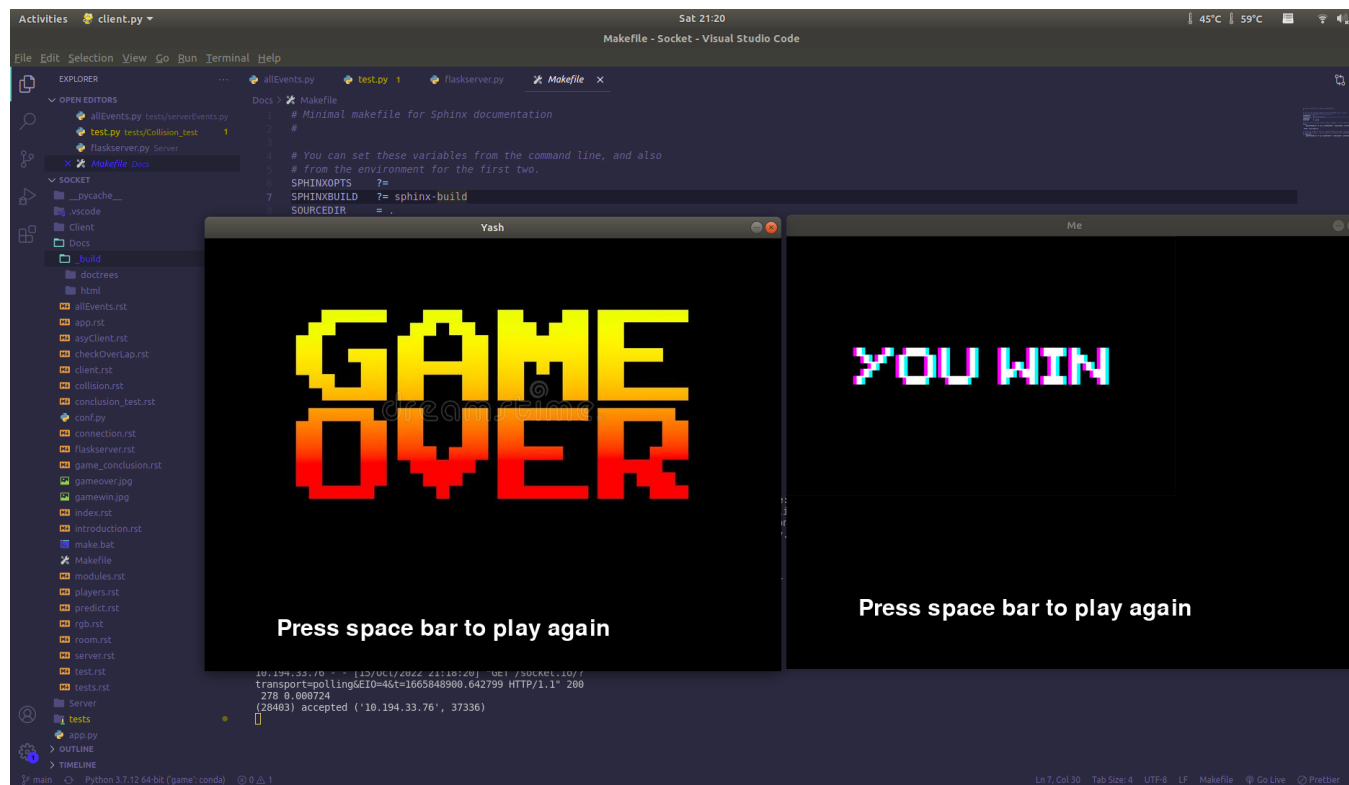

Fig. 7. Danner win situation

7.4 Escape - player wins

If player escapes the maze, win screen will show on player's screen and lose screen will show on Danner's screen.
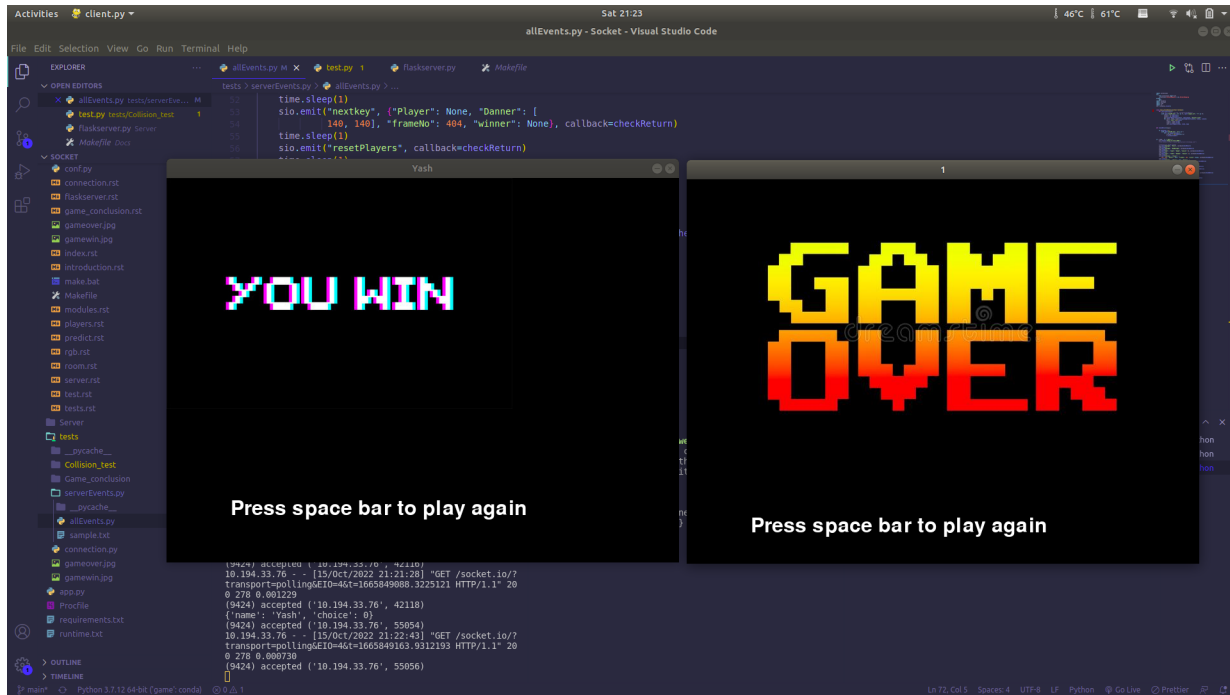

Fig. 8. Player win situation

8 TESTING

Following section describes the unit test we have written for our application.

(1) **connection test** - It is checking whether the connection is successfully established or not.

    (a) checks on successful and unsuccessful connection requests.

(2) **game conclusion test** - It is checking the win and lose logic of our game application.

    (a) check the cases where player wins

    (b) check the cases where Danner wins

    (c) check the cases where none wins

(3) **collision test** - It checks the collision functionality of our desktop game application.

    (a) checks if collision done successfully if happened that is if player and Danner gets overlapped then collision happens.

    (b) checks if collision not done if it do not happen that is if player and Danner don't overlap then collision should not happen.

(4) **server events test** - It is checking all the events that happens at the server due to the client choices.

9 ROLES

gitFollowing is the role and responsibilities distribution among the team members:

(1) **Vatsal Agarwal:**

- Auto-documentation with Sphinx: Created scripts to take the entire codebase and convert each method and class into corresponding documentation with definition of arguments

- Comprehensive report and fixed some bugs so that the game could run on different machines.

- Designed some tests using the python unittest module for the server client connection establishment.

(2) **Dhananjay Kumar Jha:**

- Designed the game GUI with the maze and players using pygame. Defined the game logic for a single system multiplayer version of the game.

- Designed custom prediction and rollback schemes that are triggered in case of high latency networks.

- System architecture design: Designed and analysed the flow of control throughout the system. Also wrote unit tests for testing game logic

(3) **Yashashwee Chakrabarty:**

- Initial design of the backend. Setup communication between server and systems using python socketio and flask server.

- Wrote logic to simulate high latency network on fast systems

- Wrote unit tests to check each event on the server with predefined outputs. Tested different versions of the backend system to look for perfect fit.

(4) **Collective Work:** The general design of the project was decided by the entire team together. The whole team floated ideas and then implemented them on consensus.

10   FUTURE SCOPE

(1) Currently our game supports only 2 players that is player and Danner. We can modify it to contain more than one Danner and player . For example a Danner has to catch more number of players to make it more challenging and fun.

(2) We can more add complex and fun mazes to our game such that everytime a new game starts, a different and fun maze is there. Currently our game supports a single maze.

(3) We can add points system to the game. Like everytime the Danner catches the player he/she gets some points for it. Player gets point for every few seconds he survive and a lot more if it escapes. At the end whoever have more points win the game. In case of more than one player the one with max player points win the game.

(4) We can add power boosters for the Danner. Lets say if there is a speed power that makes the Danner speed twice then it will help in catching players if there are many number of players. Similar power boosters can be added for the player like get invisible for a few seconds.

(5) We can make a leaderboard which will show the points of everyone who played this game before and rank them according to the points.

11 REFERENCES

(1) problem statement https://www.cse.iitd.ac.in/~narain/courses/cop701/a2.html

(2) sphinx documentation https://www.sphinx-doc.org/en/master/

(3) socketio documentation https://socket.io/docs/v4/

(4) Pygame documentaion https://www.pygame.org/docs/

(5) PyUnit testing framework https://docs.python.org/3/library/unittest.html

(6) Flask SocketIo https://flask-socketio.readthedocs.io/en/latest/

(7) eventlet https://pypi.org/project/eventlet/