Multiplayer Snake Game

Derek Ching
Computer Science
Northeastern University
Boston, United States
ching.de@husky.neu.edu

Ishan Patel
Computer Science
Northeastern University
Boston, United States
patel.ish@husky.neu.edu

Prateek Pisat
Computer Science
Northeastern University
Boston, United States
pisat.p@husky.neu.edu

***Abstract* — We will build a snake game in python using pygame library. This python binary will be controlled by an external C++ backend, which will need to handle decisions about movements of the players on the screen of the multiplayer python snake game. The C++ backend oversees determining who will be the server, and who will be the clients. Only one player can be the server. When the server received all client's messages a message handler will distribute global coordinates / collision logic to each player's snake python game.**

Keywords — Thread Synchronization, Game Design, Named Pipes, Network Communication, Python, C++.

I. INTRODUCTION

We have implemented a network multiplayer snake game, that allows multiple players to connect to a server and play the game a bit more competitively. Since the current gaming scene tends to lean heavily towards games that multiple people can play together, we, as gamers, wanted to extend a classic game to a more modern multiplayer scenario.

Our project is primarily divided into 4 major components:

- The Game
- The Backend
- The Server
- The Clients.

These components are explained in detail in the following sections.

II. THE COMPONENETS

A. The Game

We started with building a snake game, purely in Python. Python's pygame library provides a quick and easy way to implement quality games in a short amount of time and hence our snake game was built using pygame. Each player will have one instance of the game running. Each instance is a Python thread which is synchronized with the rest of the threads. We current have a support for up to 4 players playing simultaneously.

B. The Server

The server acts as the host of a game. Any player can be the server, by using the '-s' option. The server starts a socket that is bound to the server's IP Address, and is listening to the Post 8080 by default. The server is responsible for accepting connections from different players, and assigning them a player index, so as to differentiate one player from the other.

The server itself uses multiple threads to divide the functionality to different pieces.

- The Observer Thread: Responsible for checking if the server is running, every 5ms. In case the server fails, the game should end, disconnecting all the players, to avoid any socket connection exceptions.
- The Main Thread: This is the thread that is responsible for communicating with the client threads, maintaining socket descriptors for each client and handling error conditions. The main thread is also responsible for communicating with the backend for fetching the global state of the game. This game state is provided to all the clients, to ensure a consistent gaming environment.
- The Message Thread: Responsible for handing messages to and from the clients, for example, to start the game or to end the game.
- The Key Thread: This thread is responsible for communicating with the Python frontend and the C++ backend, for example, reading messages from the FIFO pipe, sending various messages back to the client, informing them that the server has left the game, or that the food/ apple has repositioned itself to a new location, or if a snake is dead or alive, etc.
- The Time Thread: Responsible for synchronizing the timing/ screen refreshes for all the individual instances of the python frontend.

C. The Client

Each non-server entity in the game is a client. A client connects to a server, and then waits for a game to start. Like the server, the client uses multiple threads to divide it functionality into smaller, manageable pieces.

- The Observe Thread: This thread checks if the client is up and running every 5ms. If, at any point in time, the fails/ crashes, that instance will be destroyed, and the said player will be disconnected from the game.
- The Main Thread: This is the main communication thread function, this in charge of connecting to the server via a socket. It's other main role is to read important messages from the server pertaining to state information about the other players, time synchronization, the state of the apple, start and end messages from the server, and message that other players are disconnected.
- The Message Thread: This thread is responsible for sending a message to the server, as long as it is running.
- The Client Key Thread: this is the thread that will control the messages between the python frontend and the backend C++, which will open and read from a FIFO pipe the messages from the python frontend and send the various messages back to the server. These messages include that the client exited from his game, that the apple has been eaten and relocated, the state of the snake (alive/dead), their id, and each snake's location, and whether the client has won or not.

D. *The Backend*

The Backend is responsible for determining which player acts as the server, initializing he server, connecting clients to the server based on the arguments passed to the executable.

III. DIAGRAMS

A. *Class Diagram*

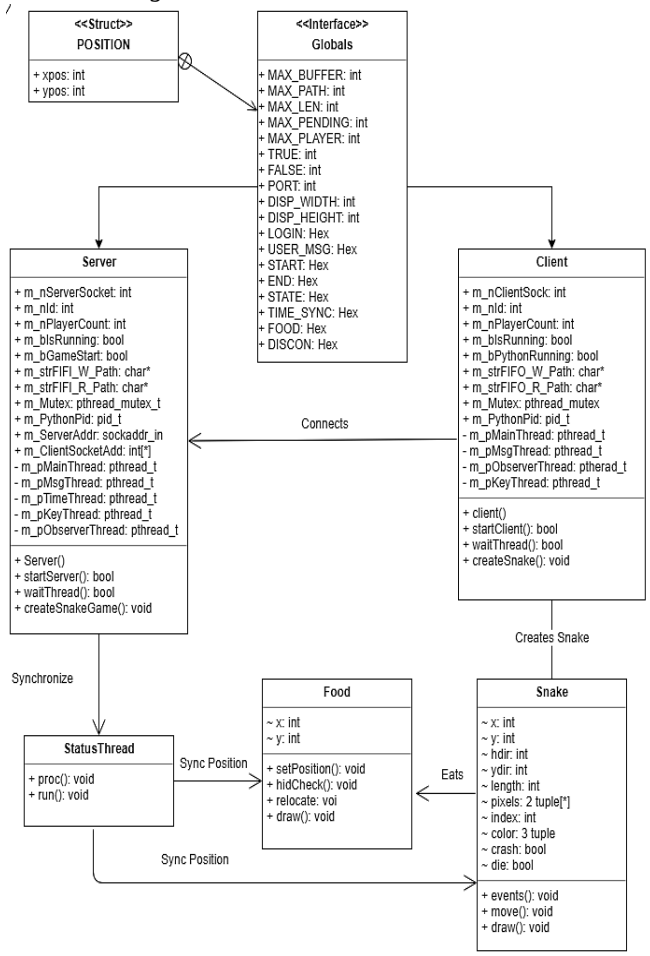

Fig.1. Class Diagram

a. Globals: This a header file that simply describes all the static attributes for our game, for example, the maximum number of players, the size of the game window, certain Boolean constants, etc. These values are stored in the back-end to reduce/ avoid using magic constants and updating certain game-wide global variables becomes easier.

b. Position: this is a struct, that is defined in the "globals" header file. This struct is used to define the position of an item, in the game, be it, a snake or food for the snake.

c. Server: This class encapsulates the server for the game. Any player can choose to be a server by using the "-s" option while running the game. The function of this class is to start a TCP server bound to the host's IP address, listening on the port that is specified in the globals header file. The server then waits for all the players to connect. (specified by the MAX_PLAYER in globals.h) Once all the players have connected to the server, the server initializes the game and sends the required game data to all the connected clients.

d. Client: Each client represents a (non-server) player in the game. The client is responsible for instantiating a snake object based on the parameters provided to it by the server, for example, the player index.

e. Snake: The snake class encapsulates an in-game snake, that is controlled by one player. Each snake object is constantly listening for key-events. On the appropriate key event (left, right, up or down) the events function will update the parameters of the snake respectively.

f. Food: The food class represents the food, once consumed, will increase the length of the snake.

g. StatusThread: This class is responsible for thread synchronization amongst all the players and ensures that all the players have consistent copy of the game.

B. *Activity Diagram*

TODO: Add Activity Diagram and

IV. TUTORIAL

Following are the steps to compile and run the game, both as a client and a server.

A. *Compilation*

We have provided a compile script that compiles the required files. The script is a bash script and may require execute previlleges to run.

Run the following command to assign required previlleges:

```
$ chmod +x compile.sh
```

To run the script, enter the following command:

```
$ ./compile.sh
```

This will create an executable file named, "snake".

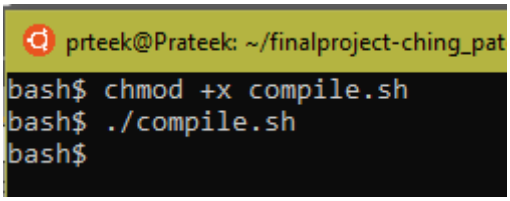

Fig. 2. Compliation.

B. *Running the Game*

As mentioned previously the game can be run with two options, viz. server and client.

If you wish to be the server, you need to run the game with the "-s" option indicating to the backend that you are the server. Note that, for other players to connect to your game, they will need to know your IP Address. To fetch your IP Address, you can use the 'ifconfig' command. Once you know you IP Address, you can start you server with the following command:

```
$ ./snake -s
```

This will start the server and will wait for players to connect. You need to wait for at least one player to join your game. The game doesn't run as a single player game.

Once you have at least one player connect to your game, you can start the game using the following command:

```
-c start
```

This will open a new window, 1 for each player connected to the game, and will continue till 1 snake remains.

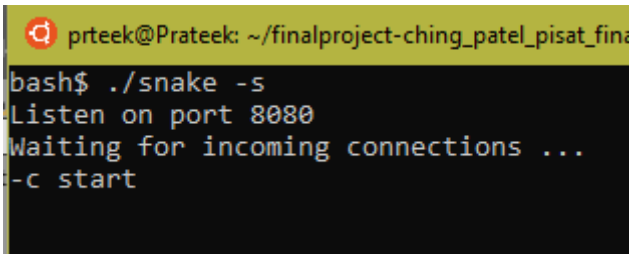

Fig.3. Server Run

If you wish to be a client, you need to start the game with the '-c' option indicating to the backend that you wish to connect to an already existing server. As mentioned earlier, you need to know the IP Address of the server. Once you are aware of the IP Address of the server, you can connect to a server using the following command:

$ ./server -c <ip_address>

If the IP Address was correct, then the server will accept the connection and a player index will be allocated to the client.

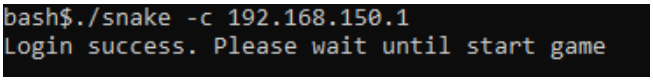

Fig.4. Connecting to the Server

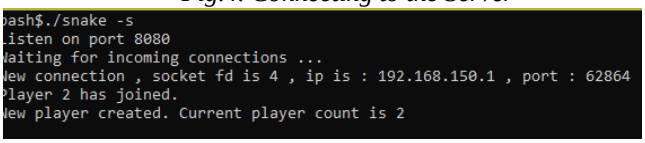

Fig.5. Server Accepting a Connection

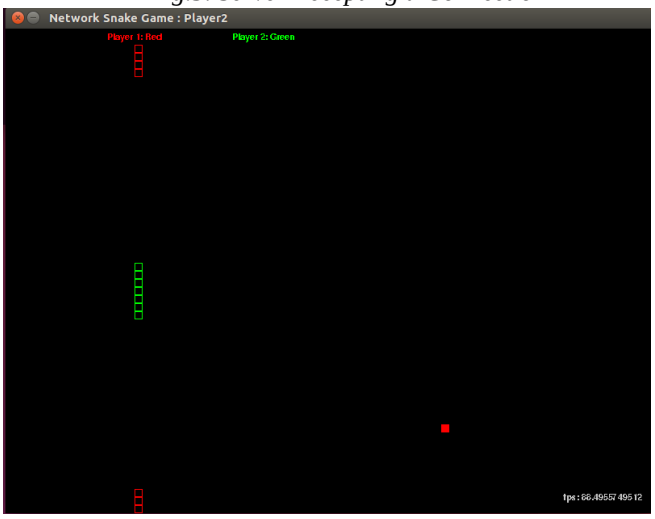

Fig. 6. Player-2 Screenshot

C. *Figures and Tables*

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation "Fig. 1", even at the beginning of a sentence.

TABLE I. TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a.] Sample of a Table footnote. (*Table footnote*)

Fig. 1. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization", or "Magnetization, M", not just "M". If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization {A[m(1)]}", not just "A/m". Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)", not "Temperature/K".

ACKNOWLEDGMENT *(Heading 5)*

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

These are the research papers and websites that we used as referenced.

[1] https://www.dreamincode.net/forums/topic/402599-computer-networking-two-player-interactive-game/

[2] https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

[3] https://www.geeksforgeeks.org/socket-programming-cc/

[4] https://www.pygame.org/docs/