Implementation of Paavo-Strike: Very Offensive

Paavo Hemmo, 014582750
University of Helsinki
Espoo, Finland
paavo.hemmo@helsinki.fi

Figure 1: Yeap, this is my game

ACM Reference Format:
Paavo Hemmo, 014582750. 2018. Implementation of Paavo-Strike: Very Offensive. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

In this documentation I will go through how I implemented the game. Mostly all the things listed in this documentation were either left unclear in the original plan or they have been completely changed. In the end I will also go through what still needs to be implemented.

2 BEFORE WE START

In order to be able to run the client, you need to download and install the module pygame. This can be done with the command pip install pygame.

3 THREADS

One thing I did not mention in the original plan was the threading. First we will look at the server side threading, then the client.

3.1 Server

The server must obviously receive packets from the clients aswell as send them to the clients. Receiving the packets from clients is implemented in the main thread, along with everything else (keeping up with the game state etc). This is because the server does not need to anything, unless it gets data packets from clients. It's only job is to modify the game state (player positions) according to the data that is received from clients.

Sending the game state packets to clients, is done in a completely different thread. This thread will send the state of the game to all of the clients, at the tickrate of 60 (60 times a second).

3.2 Client

Client side threading was a bit more complicated. Thanks to pygame (the module used) it was luckily rather simple. It is obvious that the game needs to keep rendering the data to clients at all times. But it also has to be able to receive the correct game state from the server. All this has to be done parallel to each other. In my implementation the game functionality runs on the main thread, while server packets are received in a different thread. The main thread will only render the data, that is currently stored in the client side, it does not alter the data at all. The thread that receives the data from server, will handle all the state changed in the client side, based on the data it receives from server.

4 STORING THE PLAYERS

In the original plan, I did not spend much time telling, how the players of the server will be stored. I decided to store the players in a fixed size array (size 33, the amount of players in the server). This way it was possible to always access the correct player, based on it's index in the array. For example a player with a clientID of 2, would always be located in the players-array index 2. Handling the cases, when a user disconnects from the server was also easy. If the slots 0, 1 and 2 are used (Players with id's 0, 1 and 2). If user from slot 1 disconnects, the server will simply make the element in the array's index 1 to None. So the array would still contain player0, None, player2. Now when a new client joins the server, he will be given a clientID of 1 (The first empty index). More of disconnecting from the server in the next chapter.

5 DISCONNECTING FROM SERVER

Disconnecting from the server will happen by clicking the "x" on the top corner of the game window (close the window). The user will get disconnected from the server also, if the user has not moved in 2 minutes. Other clients will get the information about a disconnected user from the type 5 packet. If a client receives data of only 4 players, and it has 5 players in it's local state. This way the client knows that 1 player has disconnected from the server, and that player will not be rendered anymore.

6 HANDLING USER INPUT

Handling the user input is done with pygame. The game will run at 60 fps, and between each frame it takes user input from the keyboard. Usually the state of the game would be immediately modified based on the input. In a multiplayer game however, the user input is first sent to the server. The server will then calculate the new state of the game, and send the state to every client. Note that the tickrate of the server was also 60, so the states in client and server-side will be in-sync well enough.

7 UNIMPLEMENTED STUFF

The threading caused some problems, that I could not solve in reasonable time. When the game is closed in client side, the thread that is receiving packets from the server will still keep running. I could not find a way to close that thread, and the loop that it runs. This is why it seems like the program just crashes when the game is closed (or when the client times out by the server).

REFERENCES