Tiny PYRPG Lab Report

Title: Tiny PYRPG by Jeffrey Smith, Chris Adams, Duncan Forsythe.

Summary:

This project will use python with the GUI library PyQt5 to create a dungeon and dragons themed game. Clients will be able to connect to a server that is hosting a game and choose their class of character. Once all players are ready, the game is started. Each player will have a turn to complete actions before ending their turn, allowing the next player to do the same. The game ends when all players have lost their health and the last player performs an action that ends the game.

Documentation:

The client starts by running the start-client.py which will open the main menu. Here the user will input a username between 4 and 26 characters long. Next, they will input an IP address to connect to, connecting to the server if it exists or returning an error. The player will be loaded into the lobby with any other players that have connected

A pre-game lobby will display all the players currently connected. In order to ready, the players must choose a profession. Anytime a player changes his profession, or a player is connected, the server updates its internal library. Players will need to press “Refresh” in order to see new players. The GUI is also updated when the player changes their profession. The clients work by waiting for buttons to be pressed which add commands to a queue. Commands such as “GET UPDATE” is sent to the server which responds with a library of the current players and their relevant information. The client will then update their own GUI. Multiple clients can be seeing different information; however, it is only cosmetic and does not affect the functionality of the game. Once all players are ready, player one can start a game. Players will need to hit refresh in order to open the game GUI.

The game will progress through two main phases. The first phase will consist of the lobby stage, where clients can change professions as well as their ready state. The next phase will consist of game stage where players can execute actions on their turn. Actions can have modifiers, which are applied immediately, and statuses, which are applied on the start of a player’s turn, can weaken over time, and have a fixed duration.

The server handles all of this in three main loops, with one auxiliary loop. The server starts by creating a new thread to handle the socket that will accept connections and join clients to the game. This is the first loop. Once a client has been validated, a new thread is started for each client. In a client thread, there are two loops. One loop handles all requests a client will send while in the lobby phase. The second loop handles all requests a client will send in the game phase. Finally, back at the start of the server, after the listener thread is started, the server enters a while loop that is checking for console input and will exit the server if and form of “exit” is typed into the console.

Network Application Protocol:

To connect players to the server, a TCP connection is established with an extra handshake function added in the client/server to verify the client was connecting to a valid server, and that the opposite was true.

The client and server use a word style command system to enact functionality. “GET ACTION”, “DO ACTION”, “ERROR”, and several others. These commands are sent to the server, along with any data they require, by sending an encoded json object over a socket. Every json object sent to the server will contain a “request” key and a “data” key. The value for “request” will always be a string, whereas the value for “data” can vary, being integers, booleans, strings, or even entire dictionaries.

The server would response in a similar format, except with a “response” key instead of a “request” key. Commands may include, “GAME DATA”, “LOBBY DATA”, “ERROR”, “JOIN ACCEPT”, and more. Json functionality is provided through the python json module.

Challenges:

The first scrapped idea was for clients being able to host games based on their IP and supplied port number. It probably could be created later, however right now the server creates a single game for 6 people. Instead, we had the server become standalone so that it could be started independent of a client.

Due to difficulties with using the python threading library along with the PyQt5 libraries, and due to time constraints preventing us from learning the PyQt5 threading and networking modules, we had to have the clients only update their cache of player information whenever they received a response from the server after performing an action, or when it was manually refreshed by the server.

Our original idea of the game was for it to be team based 1v1, 2v2, or 3v3 depending on the number of players. After the issues we had with threading and client connections we decided to just have the game run as a free-for-all. This save us time to be able to get the game to work in its current form.