Lester Ramos
5057452

Final Project Report: Paint With Friends

Introduction

The project requirements stated to implement an open ended application using real native sockets in Python, C, C++, C#, Unity Game Engine, Unreal Game Engine, or GoDot Engine. Any libraries were allowed in the implementation of the project as long as native sockets are used within some part of the application. The project requirements also stated that the program must have a centralized server to manage the multi-user/multiplayer environment, use TCP and UDP, support at least 4-10 users in the application or game with a recommended of 10 users at a minimum,
use a graphical environment, and finally, provide basic chat capabilities for the users connected in this application.

Though these requirements are clear, I came across multiple issues in the process of implementing my project. Despite not having any experience with Python prior to this class, I thought it would be a great opportunity to learn and add a new programming language to my arsenal of languages and environments. I decided to tackle on the suggest Paint Application, where the requirements go as follows: Creating a common canvas where multiple users can paint and see what others are painting, allow different objects to be painted, find a creative way to allow users to distinguish themselves from other users. After this project, I can say I have a deeper understanding of both TCP and UDP websocket protocols and I am more comfortable in using Python as one of my preferred languages.

Used Libraries

Over the course of the development of this project I was indecisive on the libraries I was using to help me build the final project but here is a list of the libraries I settled on:

- Socket: This module provides access to the BSD *socket* interface. It is available on all modern Unix systems, Windows, Mac OS X, BeOS, OS/2, and probably additional platforms.

- Sys: This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

- Json: A lightweight data interchange format inspired by JavaScript object literal syntax

- Random: This module implements pseudo-random number generators for various distributions

- Threading: This module provides low-level primitives for working with multiple threads

- Tkinter: The standard Python interface to the Tk GUI toolkit.

- ImageTK: The ImageTk module contains support to create and modify Tkinter BitmapImage and PhotoImage objects from PIL images

Implementation

The implementation of this project was straight forward, I was able to create the painting environment and chat box GUI interface using Tkinter where I was able to successfully record the users mouse movements/positions in order to draw on the canvas. With Tkinter I was also able to create dialog boxes before loading the canvas prompting users for a distinct username that was sent and stored in the server to keep track of the users connected. These dialog boxes displayed success and error messages depending on

the user's input. All the data interaction is modeled in a Python list where the command/event was the first element in the list followed by the necessary data points, for example this would be a draw command: ["paint", 234, 523, -30, 43, #F34225, 7]. These data points where sent to the server over UDP since data loss was not a determining factor, and broadcasted out to all connected users. The client then would receive this data and pass it to a data handler function where it would perform the event.

Architecture

This program was created using Python's Object Oriented features. I thought that going for an Object Oriented approach was the most appropriate model for the requirements of this program. The Server class, once initialized created the necessary sockets and ran a continuous while loop that would listen for data. The Paint class was where all the GUI features of the program where held, but used the Client class as one of its dependencies. By passing the client class as a dependency, I was able to separate all the socket logic in the client class and start a thread that is constantly listening for data while also handling all the drawing and GUI interactions in the Paint class.

Added Features

While meeting all the project's requirements, I also wanted to implement additional features. These features include:

- Circle Shape Tool
- Rectangle Shape Tool
- Random Color Assignment
- New Canvas Tool
- Historic Data Saving

These features are self explanatory, the circle and rectangle tools are used to create the respective shapes. The random color assignment is what distinguishes all the the users from each other and is assigned at the start of the program, the chat box displays a welcome message with the user's color and is also broadcasted to all other connected users in order to notify that a new user has joined the drawing session. The new canvas tool allows users to wipe the canvas from all drawings and start on a fresh new canvas. The Historic Data saving feature is one of the most important features of the program which saves every single command sent to the server. The server then sends this history data to every new connected user over a TCP socket before initializing the GUI in order for the users to sync all of their drawings. With this historic data the drawings are synced and if a new user joins while existing users are already painting, they can see what other users have painted and collaborate together making it an overall better experience for all users.

Results

I am overall pleased with the suggested Paint Canvas project. It strengthened my understanding of socket programming as well as strengthening my programming skills in Python. The hardest issue that I faced during the development process was using both TCP and UDP sockets to receive the history and the real time data. The history data was too large to send over UDP and I would've needed to implement my own functions in order to handle the splitting and sending of the data. Initially, when just adding a TCP socket and listen for history data caused my program to experience a lag and the GUI was freezing up. In order to solve this issue I sent the history data over TCP right when the client program starts before loading any GUI. After receiving the history data I closed the TCP connection and socket, thus reducing the lag and delay in the GUI.