HW1 Two-Player Online game

112550151 周佳瑩

System Architecture（Lobby、P2P、DB）

Lobby

The main library I used to open connections is asyncio. The lobby server is hosted on the IP and port set in config.py via asyncio.start_server., once started, it loads a list of users from userdata.json. It listens to all the inputs from the clients with the function handle_client, but doesn't allow the user to do anything without logging in. The lobby server also constantly keeps track of the clients' status, when a client logs in, creates a room, joins a game, or logs out, it updates the online_users dictionary in the lobby server. The lobby also broadcasts a list of online users whenever a user logs in, logs out, or disconnects. The server avoids overwriting data by using "async with [user/online_users/room]_lock", this makes sure that no two sources are writing into the dictionaries at the same time.

When a user logs in, they first receive a message notifying that they successfully connected to the lobby via TCP, and a message telling them which UDP port they connected to. Then a list of available commands is shown:

Available commands:

register <Username> <Password> - Register new account

When a user registers, the lobby server checks if their username is available, then the password is hashed via sha 256 and saved into the server class, and updated into the userdata.json file.

login <Username> <Password> - Log in

When a user logs in, the lobby checks if their username exists, then checks if their hashed password is the same as saved in the server, if all is correct, the user's status is changed to idle, and their username is added to the list of online users.

logout - Log out

When a user logs out, their name is removed from the online_users dictionary, and any rooms that they have created are deleted.

create - Create room

This allows a user to create a room to invite another user into, to play a connect 4 game with.

invite <Port> <Room ID> - Invite user to join room

I'll address this in the P2P section.

exit - Leave client

This logs the client out and closes the connection.

help - Displays list of available commands

This displays the list of available commands on the client and doesn't require the lobby server to respond.

status - Displays current status

This calls the lobby server to display the list of online users and available rooms.

scan - Scans UDP ports for available players

This is executed on the client and scans all the available UDP ports set in config.py to see if any other UDP connections can be made aside from the user's own. If a connection cannot be made, it is assumed that there is another player available to connect to. The result of scanning is either no players found, or a list of available ports is provided.

P2P

For a player to invite another player, they first have to create a room to invite the other player too, then they have to scan all available UDP ports to see if there's a player online. When an invite command is sent, the server first checks if invite is valid – whether the user and room are valid, then it notifies the client to send an invite to the other player's port via UDP. The UDP listener on the other player's client asks them whether to accept or decline. If the invitee declines, the decline message is sent to the inviter via UDP and nothing happens. If the invitee accepts, an accept message is sent to the inviter via UDP, and a join room message is sent to the server, which will send a message to the inviter to start a game server and start the game.

Database

The data for the server is a server class defined in config.py, this server saves the users, rooms, and online users, locks, and is used across all files. While the list of rooms and online users is only saved in the server class, the list of users and their hashed passwords is saved in userdata.json.

The server and all clients log to logger.log, which notifies the administrator of any actions made by clients, including registration, login, creating rooms, inviting and such.

Communication

The detailed information on communication between each player and server（through JSON, ByteString, etc.）

Communication between the client and server are in JSON format.

Messages from the client to the server are commands in JSON format, these commands contain a command part and a params part, the server executes the command from the command part, and uses the parameters provided in the params part.

Messages sent from the server to the client is a JSON contain a status and a message, the client executes different actions depending on the content of the message.

For invitation, the message sent from player A to player B and vice versa are both in JSON, and the receiver responds depending on the given information. During gameplay, the move made by the opponent is also a JSON, containing only the column the opponent chose, and the gameboard is updated locally on each client.

Game play

The game I implemented is connect 4, it uses a game board with 7 rows and 6 columns, with one player playing as X, and the other as O. The players take turns dropping their pieces, choosing which column to drop pieces, the pieces cannot be placed under each other and can only be dropped on top of the previous pieces. The first player to drop 4 pieces on a line wins, meaning the first player with 4 pieces connected either horizontally, vertically, or diagonally wins.