GAME HUB - READ ME

Amanda Erika Lim, 2014-11126
Aidee Criselle Peñamora, 2015-13983
Angelique B. Rafael, 2015-02452

December 9, 2019

Contents

1 Game Hub

This section states how the game hub works. It has been decided that the game hub will be a server-client system instead of peer-to-peer. It is because the player/client should not have direct access to the main code, or the games themselves, which would have been the case if a peer-to-peer hub was implemented. A server-client system ensures that the client side is disjoint from the server, with the code consisting mainly of send and receive threads.

It is assumed that the Game hub handles any possible error done by the connecting client is to kick them. It is assumed also that once the game starts, there's no more players connecting to the room or disconnection doesn't happen.

1.1 Client

Code to be run by players. Includes threads for sending and receiving messages from the server. Does not directly interact, in any way, with other clients. If the client is the host - the one who created the room, only they can start the game by sending the message *$$game$$*. It has a clear function that is equivalent to the command *clear* of the terminal. This is called by the server by sending the message *$$clr$$*. The client can only exit if the process is terminated (e.g. Ctrl+C), and if it receives the *$$quit$$* message from the server.

1.2 Server

The server deals with incoming players, room creation, putting the players to the right rooms, and ensures the game playing in one room do not affect the others. It prints out all those who connected and disconnected. The accepting of incoming connections is done in a stand-alone thread. Its role is to keep checking if there's a process wanting to

connect and if there's a process that connected, it will create another thread to handle the client. The function name of this is *accepting_incoming_connections()*. The handling of the client includes getting the username, creating the rooms, connecting a player to an existing room, and playing of games. The function name is *handle_client()*. The exclusion of games from one room to another is done using the Room class, which is stated below. The games also use local variables to ensure the exclusion. The Room class variables are saved in the variable *hotel* as stated below. If the user inputs a password that doesn't exist in the *hotel* keys, then the user will be kicked out.

The server initializes the following variables at the start:

1. username - dictionary of usernames (key: username; value: client socket)
2. clients - dictionary of clients (key: client socket; value: username)
3. addresses - dictionary of client addresses (key: client socket; value: client address)
4. hotel - dictionary of virtual rooms (key: password; value: Room variable)
5. HOST - IP address of the server
6. PORT - port number (fixed to 6969)
7. BUFSIZ - buffer size (fixed to 1024)

The Room class is defined as such:

```
# ==== The struct of the room ====
class Room(object):

  def __init__(self, name, username, game, password):
    super(Room, self).__init__()
    # Initialize the variables first
    self.names = []
    self.user = {}
    self.game = ""
    self.password = ""
    self.num = 0
    self.state = False              # If playing or not

    self.names.append(name)         # List of players (client variable)
    self.user[name] = username      # List of usernames of the players
    self.game = game                # Name of the game to be played
    self.password = password
    self.num = 1;                   # Number of players in room

  def add(self, name, username):
    self.names.append(name)
    self.user[name] = username
    self.num += 1

  def datingsim(self):
    # Initialize the variables needed for dating sim
    self.place = []
    self.taken = []
    self.choice = 0
    self.score = {}

    for i in range(4):
      self.place.append("")
      self.taken.append(False)

    for i in self.names:
      self.score[i] = 0
```

2 Games

This section describes in detail how the games work. The games have their own code which are imported inside the server code.

2.1 Multiplayer dating sim

"My Last Days with You - Dating Sim" is a competitive dating sim where the players flirt over one character and the one with the highest affection points will win. The plot takes place within 7 days with the 7^{th} day as the game's ending. It can only handle a room with at least 2 players and at most 4 players. It has 4 places where the events can occur and that the players can exclusively get. In these 4 places, players have 3 choices to make: 2 known choices, and a time out choice. For every choice made is equivalent to 3/5/7 points. The game uses the variables in the Room class.

The text of the game is sent to the players with certain delays. Unlike known dating sim games which has the player press a button or key to forward the story, this game does it on its own. It is only during the choosing of places when the game waits for the player to make their choice that the text stops printing. The other waiting for player input is the event decision which has a time limit of 30 seconds. The player is assumed to give smart inputs.

The functions included are:

- gamecast() - broadcasts the message to all the players in the room
- text() - sends game text to the player
- clear() - sends the *$$clr$$*
- init() - initialize the room dating sim variables
- dayx(), homex(), parkx(), schlx(), and libx() - story of the game where x = {0, 1, 2, 3, 4, 5, 6, 7}

The following features are implemented as such:

- Time limited decision-making - Using the ***.settimeout(****value****)*** of the socket API, such that the *value* is set to the number of seconds it takes for the *recv()* function to wait for inputs
- Exclusivity - See the Room class explanation in 1.2 Server
- Scoring - See the *datingsim()* function of the Room class in 1.2 Server
- Tie-breaker - Whatever the function *max()* picked will be the winner

2.2 BFF Test

The BFF Test is a quiz game played by several players. There are two stages:

1. Group stage
    - There are 5 questions pertaining to the players.
    - After each question, players will be given a maximum of 30 seconds or until everyone gets to answer.
    - By then, an extra 10 seconds will be given to finalize the players answers; the last answer of the player will be their final answer.
    - Answers should refer to a player, else, the answer will get no point.
    - Those who answer similarly to most players will get a point; In case of a tie, everyone who's part of the tie will get a point.
    - Note that basic typographical errors will be considered, so don't worry about having typos.

2. Solo stage
    - Contrary to the stage name, this is still played by group.
    - There are multiple rounds in this stage, each highlighting one player.
    - Per round, there are 3 questions about the highlighted player.
    - After the answering portion, which has the same mechanics as the Group Stage, the highlighted player will be given 30 seconds to grant points to other players if he/she finds their answer acceptable.
    - In the case that the highlighted player leaves or gets kicked, the round ends regardless if the 3 questions are done or not.

The player with the most points win.

Other general rules:

- Commands starts and ends with a ”$$”. Some commands are:
    - *$$ans,<answer>$$* - to answer
    - *$$kick,<username>$$* - to kick someone out of the game; note that you can’t kick yourself out
    - *$$quit$$* - to quit the game
- There are other commands available depending on the state of the game. They will be mentioned in the game during the corresponding states.
- Messages without the opening and closing ”$$” are considered normal chat messages and are broadcasted.
- In the case that only a single player is left, the game will end immediately and the player wins.
- Also, take note that it is assumed in the game that the players will send a last message, so please do.

2.3 Who Wants to be a Millionaire?

”Who Wants to be a Millionaire?” is a single-player game based on the famous television game show of the same name. It is a trivia game, where the player is asked general knowledge questions and they have to choose between four choices within a given time limit. There is a total of three rounds, each composed of three levels. The time limit is 15 seconds for the first round, 30 seconds for the second round and 45 seconds for the third round. There is also a ”lifeline” in the game play, which are basically hints for the player when answering. The game ends when either the timer runs out before the player answers, when the player gets an incorrect answer or the player has finished all of the three rounds.

The game includes a main menu feature, where the player can choose to start the game, ask for how to play or exit the game. This will be the default screen when the player enters the game or after a game is over.

The functions included are:

- menu() - this is the default screen where player can choose to start the game, know how to play, or exit the game
- help_menu() - shows how to play, goes back to the main menu afterwards
- get_ans() - gets the answer from the player and determines whether it is correct or not
- lifeline() - shows the player which lifelines are available for use and gives a hint based on the lifeline
- play() - the function that generates the randomization of questions and determines the round of the player
- clear() - sends the *$$clr$$* to the player
- text() - sends the game text to the player

3 Getting Started

3.1 Prerequisites

Must have Python3 installed.
Follow this guide on how to install Python3: https://realpython.com/installing-python/

3.2 How to Run

3.2.1 Server

1. Download and extract the "Game Hub" ZIP file found in the GitHub repository.
2. Open the terminal where the files are extracted from.
3. Determine the IP address of the Wi-Fi or local connection. Use the command *ifconfig* for Linux and Mac terminal, or *ipconfig* for Windows command line. It is the *inet* in Linux and Mac while it is *IPv_Address* in Windows
4. Run the server code by entering this command in the terminal: *python3 server.py*
5. Input the IP address that was gotten in step 3. This will be the server IP address.
6. Let it run.
7. Close the server by pressing *Ctrl+C* in the terminal.

3.2.2 Client

1. Download and extract the "Game Hub" ZIP file found in the GitHub repository.
2. Open the terminal where the files are extracted from.
3. Run the client code by entering this command in the terminal: *python3 client.py*
4. Input the IP address of the server that was gotten in step 3 in 3.2.1 Server section.
5. Ensure that one of the clients creates a room in order to play.
6. Play to your heart's content.