BOĞAZIÇI UNIVERSITY

Online Bingo Game

SWE 544

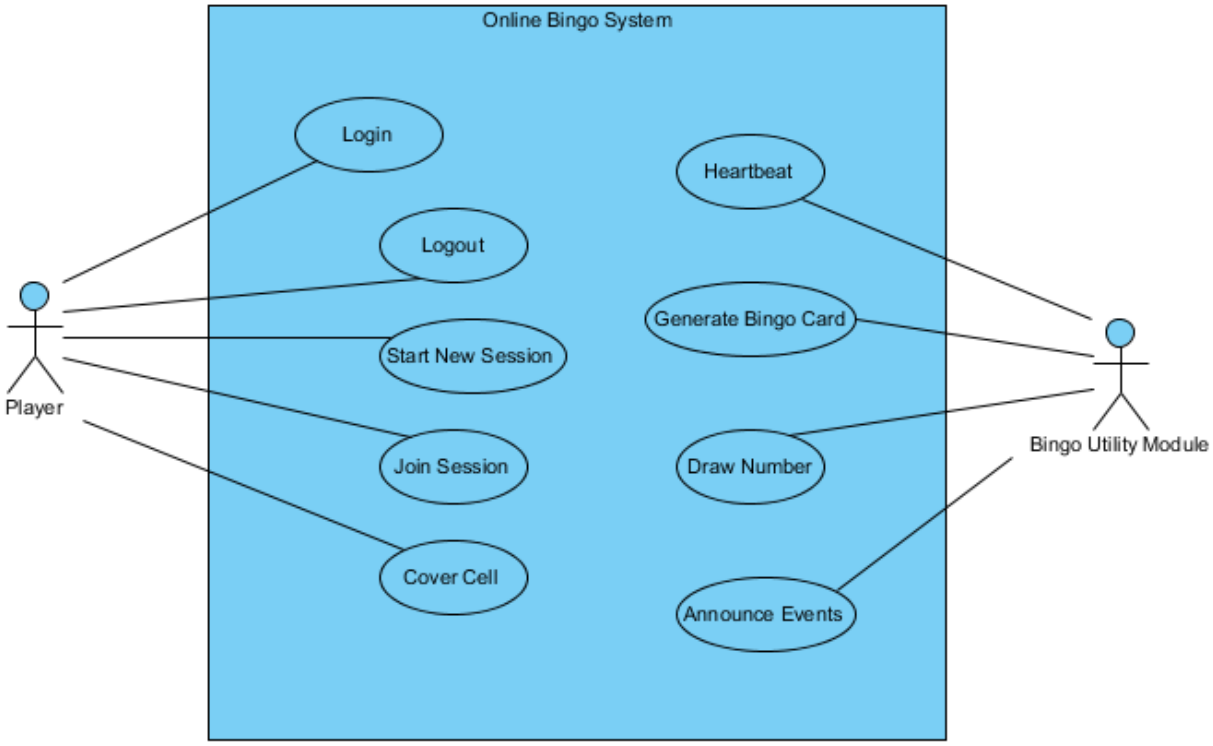
Talat Aydın Çıkıkcı

Introduction

"Online Bingo Game" system is a multiplayer game of bingo where each player can connect to a server using a username and play bingo with other bingo enthusiasts. System consists of a server that hands out the bingo cards and draws numbers and clients that receive drawn numbers and match them with fields on their designated bingo cards.

System is coded in Python 2.7.11 and has a very simple command line GUI that allows some basic tangibility to the bingo card for the users.

Use Cases



There are two main actors in the system. The "Player" and the "Bingo Utility Module". Each actor has a different set of ways to interact with the system.

Player Use Cases

Player actor is the client that connects to the server to play.

1. **Login**
    - In order to play, a player first needs to log into the system. To achieve this, login message is sent to the server accompanied by the player's desired username.
    - If the username is available the player is logged into the server and the username is assigned to that player.
    - If the username is already taken then, the login will be rejected and the player will be asked to login with another username.

2. **Logout**
    - When the user exits the application, a logout message will be sent to the server, which in turn ends the user's connection and then releases the username to be used by other players.

3. **Start New Session**
    - After a player has successfully logged in, it has two choices. First one is to create a new session, and the other one is to join an already existing session.
    - When user attempts to create a new game session (game room), the client sends a new session message to the server. Server then allocates the necessary resources to keep track of the users in the session, their game cards and the session time.
    - Session creator can leave the session, and the session will not be terminated.
    - A session will be terminated if there are no players in the session.
    - A session has some requirements before it can start.
        i. It requires a minimum of 2 players.
        ii. It has a maximum player limit.
        iii. Session will wait for a predetermined amount of time before the game starts, until maximum player limit is reached. If timeout happens before required number of players have joined then the game session will be terminated and players will go back to "New/join session selection".

4. **Join Session**
    - After a player has successfully logged in, it has two choices. First one is to create a new session, and the other one is to join an already existing session.
    - Players should be able to see all the game sessions that have not started yet.

- Players should be able to see all the players in a session that has not started yet. This is done by a query to the server.
- Players can go back to session selection page.
- Players can choose to be “Ready”. When a player chooses to be ready a ready message is sent to the server. If the conditions are favorable and every player has sent a ready message, the game will begin.

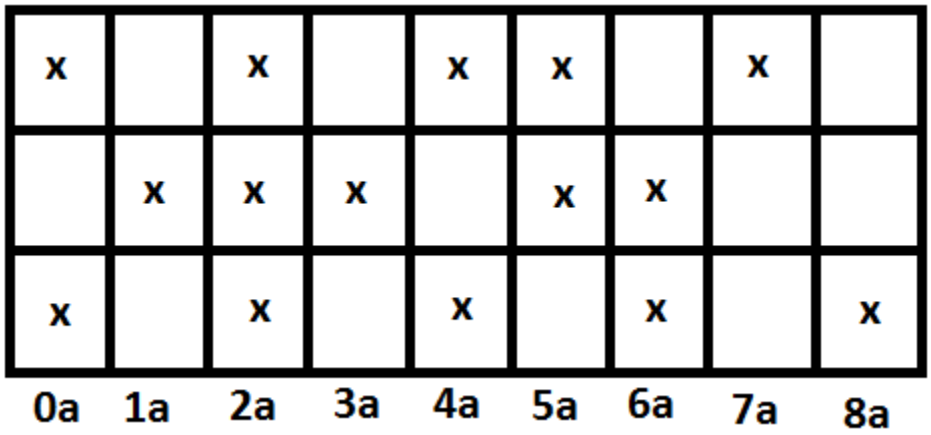
5. **Cover Cell**
    - Server sends a random number to the users each turn, when every user sends “OK” message to the server.
    - If a number is received by a user that exists on their bingo card, they can cover it. When user wants the cover the number a message is sent to the server that the player did so.
    - If a line is covered completely, it is called a “cinquina” or a “cinko”. This is tracked in the server and announced to all players by a message.
    - If a user does three cinkos, It is called a “bingo”. This is also tracked at the server and announced to all users. In bingo is done by a player, that player wins and the game session ends.

Bingo Utility Module Use Cases

Bingo utility module actor exists in the server and manages some serer operations.

1. **Heartbeat**
   - Bingo system periodically sends a heartbeat message to the clients to see if they are still available.
   - Clients should respond with a keep-alive message to inform the server that they are still active.
   - If the keep-alive is not received by the server for a predetermined amount then the server terminates that player's connection and makes the username available to other users.

2. **Generate Bingo Cards**
   - At the beginning of the game server sends every player their bingo cards with randomized numbers from 0 to 90. Bingo cards consist of three lines of 5 numbers.



   - First column has numbers 0-9, second column has numbers 10-19 and so on…
   - Module generates a bingo card for each player and sends it to the related player. Player responds with an acknowledgement message when the card is received.

3. **Draw Number**
   - Each game session has a "Bag" that contains one of each number from 0 to 90.
   - In each game turn, the module randomly procures a number from this set and sends it to each player.
   - After the number is sent it is taken out of the set.
   - Players send an acknowledgement message to indicate that the number has been received.

4. **Announce Events**
    - The module sends and “cinko” or “bingo” event in the game to all other players in the same session.

Client messages

Operation	Message	Parameter	Response	Parameter
Login	LOGIN	username	LOGINOK	userName
			LOGINREJ	
Logout	LOGOUT	username	LOGOUTOK	
Create Session	CREATESES	gameName:maxPlayer	CREATEOK	name
List Sessions	LISTSES		n * SESSION	sessionName:(user1;user2;…)
Join Session	JOINSES	sessionToJoin	JOINOK	
Leave Session	LEAVESES		LEAVEOK	
Cover Tile	COVER	Number	COVEROK	
Cinko	CINKO	username	CINKOOK	
Bingo	BINGO	username	BINGOOK	
End Turn	ENDTURN		ENDTURNOK	
Cover Nr.	COVER	nrToCover	COVEROK	
Heartbeat	TIC		TOC	
Error	<invalid>		ERR	

Server messages

Operation	Message	Parameter	Response	Parameter
Send Number	DRAW	number	DRAWACK	username
Announce Cinko	CINKO	username	CINKOACK	
Announce Bingo	BINGO	username	BINGOACK	
Send Bingo Card	CARD	username:(row1,row2,row3)	CARDACK	
Heartbeat	TIC		TOC	
Error	ERR			

ServerThreads

There are several different threads in the system that allow the system to operate. Below is the list of them and their purposes.

1. **BingoServer**
    - BingoServer thread is the main thread that handles the incoming messages and the outgoing messages to the server side.
    - It creates the socket that will be used by the client to connect to the server and listens to it using a select() system call.
    - It is the “main function” of the bingo system. It generates and destroys all the other threads that are used in the system.

2. **ReadThread**
    - ReadThread thread is generated once by the BingoServer thread just before the server starts listening to the network.
    - BingoServer thread passes the incoming messages directly to this thread by utilizing a Queue. ReadThread thread continuously scans the “readQueue”.
    - This thread parses the incoming messages to obtain the command messages and the parameters if there are any.
    - It then does the actions that are relevant to the received command message. These actions include generating new game threads.
    - After the command message is processed if there is a necessity to return a response to the sender, it places the response message to the send Queue.

3. **WriteThread**
    - It continuously scans the send Queue, which is called the “writeQueue”.
    - When an entry is “put” to the writeQueue, this thread forwards it to the socket so that it can be sent to its destination.

4. **GameSession**
    - It is generated by the WriteThread, if the client sends a create session message.
    - It can be created many times, since the server supports many game sessions concurrently.
    - It first waits for timeout OR maximum player limit to be reached. If timeout occurs session is terminated, if max player limit is reached game begins.
    - When the game begins, this thread creates unique bingo cards for each player in the session.
    - After cards are distributed to the players, this thread generates a random number between 1-90 and send it to the players and waits for either turn timeout or “end turn” request from every player.

- Game events are all handled in this thread, almost-independent of the game client. Game client's main purpose is to display information to the players and give them a feeling that they are actually playing. But GameSession thread can progress from game beginning to the game ending itself as long as there are at least two players in the session.
- After a number is drawn, game session automatically "X"'s the numbers in player's cards, but it still waits for player confirmation before sending them the card with the new "X".
- Similarly when a "cinko" event happens, thread tracks it but does not announce it to the players if the player that has done a "cinko" sends the message to do so.
- When "bingo" event happens, it progresses similarly to the "cinko" event. But if the player misses to call "bingo", he will be notified again in the following turns that he actually has a "bingo".
- When a player acknowledges a "bingo", all players in the session are notified and the game thread terminates.

5. **CountdownClock**
    - It takes an integer value while it is created. This integer is used as the starting point of the countdown.
    - It counts down to 0, by reducing the initial value by 1; pausing for 1 second before every reduction
    - It's "countdown" parameter is initially False. When it reaches 0, it becomes True.

ClientThreads

There are several different threads in the client, contained in a single file. Below is the list of them and their purposes.

1. **BingoClient**
    - BingoClient thread can be considered the "main function" of the client side.
    - It generates all three threads that operate in the client side.
    - It connects to the BingoServer, using manually entered parameters.

2. **ReadThread**
    - ReadThread thread is generated once by the BingoClient thread.
    - It operates just like the ReadThread thread in the Bingo Server. One big difference is that, aside from communicating with the WriteThread, it also communicates with the SessionDisplayThread.

3. **WriteThread**
    - It continuously scans the send Queue, which is called the “writeQueue”.
    - When an entry is “put” to the writeQueue, this thread forwards it to the socket so that it can be sent to the server.

4. **SessionDisplayThread**
    - This thread generates all the content that is seen by the player.
    - It prints necessary information to the command line, as well as wait for inputs from the player in the command line.
    - It translates the inputted commands and places the related command messages to the writeQueue.
    - Even though BingoClient is the most fundamental thread that spawns the game client, SessionDisplayThread is the thread that the user actually interacts with.

Conclusion

I have learned a lot about sockets, python, multiprocessing and threading during this project. I have gained insight about protocol messaging. These are all my gains from the project. The resulting code however has some flaws and is not exactly like the expected product. In this section I will list the accomplishments and failures of the project.

Accomplishments

- Server was implemented using “select()” system call. This allows a single thread to listen to the socket very fast, and take necessary action when a message is received. I have tested the system with up to 12 clients, which could send messages to the server and receive responses without any concurrency issues.
- System generates unique bingo cards for every player in the game session, with 3 rows and 5 columns, each row containing only one of a tenths group, sorted from smaller number to the large and each card in conformance with a real bingo card.
- Client side can display all the necessary information to the player through the command line in a clear and understandable way.
- Fail cases are mostly handled in the code. Invalid entries by a user is tolerated by explaining the issue and asking for input again, system exceptions that are generated due to system environment are mostly handled so that the system does not crash if an exception is thrown.

Failures

- Most fatal issue with the system is that I tried to implement a system that could handle many game sessions, but I could not distribute players to different game sessions. I did not have time to figure out how to send game session broadcast messages ONLY to the players in that session. Since currently there is no player distribution, attempting a usual flow of creating and playing a game cannot go further than actually creating the game thread and waiting for other players. So if you wish to test functionalities that take place AFTER a game session begins, you would have to do so by using a dummy game session.
- Also in relation with the previous comment, I have figured out that distributing players to game sessions and broadcasting messages to players in the current game session can easily be done by creating a connection thread for each player connection that is accepted. As I have mentioned also in the previous comment, I did not have to change the server implementation.
- The system is missing some implementations, since most of the effort was spent on the first comment. These implementations include; send drawn number to player, “bingo” check, “cinko” check and drop player after heartbeat timeout.
- There are some unhandled exceptions that may cause related threads to become unresponsive. This can be resolved by a little more testing and debugging.