CPSC 441

Assignment 3

Fall 2019

Title: C++ Tic-Tac-Toe Network Game

Name : **Gregory Ord**
UCID: **30020595**

Name: **August Sosick**
UCID: **30050299**

Name: **Nathan Darby**
UCID: **30033588**

Name: **Grigory Devyatov**
UCID: **30028553**

Tutorial Section: **T01**

Professor: **Mea Wang**

Teaching Assistant : **Asif Ali Mehmuda**

Date : **2019-11-08**

Group: **#5**

Room: **ST 064**

Network system overview:

The system will consist of one server, serving up to a predefined number of clients connected. Some clients may be in game while others setting up waiting rooms or just waiting in the main lobby.

The client will always interact with other users (playing and chatting) through the server. Outside of game play, the client will simply be served content. During games, on the other hand, the client will have the game state available to be able to check validity of input to reduce demand on the server.

The system will also have a small backend application to allow for admin to ban users without turning off the server.

System analysis:

The system tolerates delay. Both the game play and the supporting features are not very time sensitive. Any delay less than a second can be considered negligible, because it's a turn based game.

The system doesn't tolerate loss. Every packet will contain the necessary information to play the game and otherwise interact with the server. A packet lost during a game will put the players and the server out of sync. For example, server will be waiting for a move from one of the players, while the player will already be waiting for game state update after the opponents move.

Based on the fact that the system can not tolerate loss, we chose to use TCP for all network communication of the system.

Use cases:

Client application use cases:

Login:

After starting the client application and establishing a connection to the server, the client will prompt the user for a login. This will either lead to a successful login or an error message and a prompt to try again. A successful login will put user in the main lobby of the game. Once the user logs in, their status will change to online.

Reconnect:

If when a user logins and the server sees it's in one of the ongoing games, instead of putting the user in the main lobby it will reconnect the user into the game.

Main Lobby:

In the main lobby the user will have an option to see leaderboard, create a waiting room, join a waiting room, delete a waiting room or observe a waiting room.

Leaderboard:

Asking for the leaderboard will display the standings of each registered account, based on their win/loss/draw record.

Waiting Room:

Each waiting room can contain at most two players and at a maximum of predefined number of observers. The players inside the waiting room have the option to toggle their ready status. Once both players are ready in the waiting room, the server will begin a game. Inside the waiting room the players also have the option to leave or chat with each other. The observers inside the waiting room have the option to leave it.

Game Play:

Once both players indicate they are ready, they are placed in the game with observers also able to look at the game state. The player will alternate taking turns making a move in tic tac toe. Once it is their turn they will be prompted to enter a row and column to put their mark

on. When a valid cell is selected the cell will be marked and the players turn will end. If the user ignores the prompt to enter in their data (30 second timeout), they will forfeit the match.

In addition to the main game feature the players also have the option to message the other at the end of their turn. These messages will only be visible to the two players and not spectators. They can also forfeit the game at any time, resulting in their loss.
When a win or a draw is reached, both players and spectators will be shown the result. The players will then have a choice between rematch (which will exit the game mode and put them back in the waiting room) or to quit (which will exit the game mode and return them to the main lobby).

During the game, spectators can only see the game state and their only option is to exit the game which will also return them to main lobby.

Server application use cases:

The server side also needs a small interface to allow to register new users, list all registered users and ban users. This will be available in the server application right after the server starts and creates a server socket.

List all registered account:

Lists all registered account, with their ID and win/loss/draw record.

Register new user:

Prompts the user for a username and password. Once a valid, available username is given the server displays a success message and returns to the main screen.

Change password:

Prompts for a user name or ID. Once an existing user is identified, prompts for a password.

Ban user:

Prompts for a valid username or ID and bans that username from logging in.

Feature list:

Basic Features:

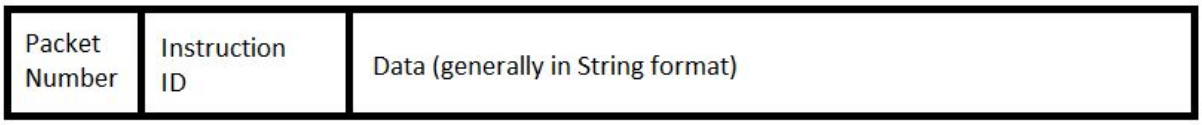
- User login/logout
    - Users will be able to login and logout of their accounts by their username
- Interactive game playing
    - After users have been matched they will commence playing the game against each other
- Server checking game movements
    - The server will check game movements to ensure the game is played fairly
    - Server checks if there is a winner and send messages to the players to end the game

Additional Features:

- List of users
    - One of the options in the lobby will be to list all users and their active status
- Banning users from the game room
    - A user may have their account banned from the server
- Leaderboard for all users
    - A leaderboard for all users will be kept and updated, viewable from the lobby
- Observers to the game
    - Users will also have the choice of observing active games
- Joining, creating, or leaving a game room
    - Users will have the ability to join an existing game room, create a new game room, or leave the game room they are currently in
- Game rooms and lobby
    - There will be a main screen displayed when a user logs in, giving them options of where to go in the lobby
- Assigning a winner points
    - assign players to their respective marks, and add the winning players points to their account
- Multiple Game Rooms
    - Enable multiple game rooms and user to be playing concurrently on the same game server

Message format:

To insure that the message format used to communicate data between client and server is as short and effective as possible we are going to utilize a simple prefixed message design as laid out in the diagram below:



The message is sent entirely in a string and assembled at the client and server side through string appends. Packet number is 3 char's at the front of the string indicating to order in which to assemble the data if the message was too large to send in one packet. The main function in both client and server utilize the case function inside a continuously looping main function and the variable input for the case function is provided by the instruction id within the message. That will tell the client or server what the other wants to do while also telling it what methods to run in order to insure it can do what is asked of it. We are not using different message formats to send a variety of messages, we are using a multipurpose message format that differentiates itself and where it should be used by the instruction ID provided in the front of the message. An example of the message a client would send to log in would be: "000_000_myname mypassword\n". Message parts are separated by "_" so the data can easily be split according to its type by the person receiving it.

Server:

- Login: Server replies with data "1" if the user was logged in (along with all current data the client need such as the current game rooms and availability), "0" if failed to login in, or a string with the data about the current account if a user is switching devices. Instruction ID: 000
- List: Server returns a string of all users, while filtering out those who are not active. Instruction ID: 001
- Create: Server creates a new game room and replies with data "1" if successful, or "0" if unsucessful. Instruction ID: 002
- Delete: Server deletes a game room and replies with data "1" if successful, or "0" if unsucessful. Instruction ID: 003
- Observe: Each time the server variable that contains a live copy of the board is updated, it's details are sent to a user who requested them. Identical format to the send board message below, except with a different Instruction ID since the methods needed are different. Instruction ID:004
- Leadboard: Server replies the with the local variable that contains the most recent leaderboard standings. Instruction ID: 005
- Exit: A server will log out the user who sends this as well as terminate their connection and update their flag to offline. Server logs a user out and replies with data "1" if successful, or "0" if unsucessful. Instruction ID: 006
- Send Board: The server will send the latest copy of the board on its local server to the client that requested it. Additionally, note that it is the board variable that has the

messages individuals send to each other appended on the end. That is how messages are sent to each other. Instruction ID: 007

- Recieve player choice: After having received the client's choice on their next move, the server can reply with a "1" indicating that the user's move was valid and their turn has now ended, or with a "0" indicating that their move is invalid (this subsequently prompts the user to resend a corrected choice). Instruction ID: 010
- Game completed: If at anytime the server send this message to the clients, it indicates that game has ended. The packet's data contains the name of the individual who won and if the data contains the work "stalemate", the game has ended in a tie. Instruction ID: 011

Client:

- Login: Client sends a request to the server to log in by providing their ID and password in the data section of the packet (separated by spaces). Instruction ID: 000
- List: Client sends a request for a list of all available users. The data field in this message is left blank/ignored. Instruction ID: 001
- Create: Client sends a request to the server to create a new game room by providing the name of the new game room in the data section. Instruction ID:002
- Delete: Client sends a request to the server to delete a game room. The data section contains the exact name of the room the client wishes to delete. Instruction ID:003
- Observe:Client sends a request to the server to update it with the current status of the board in a specific game room. Client does this by providing the exact name of the game room in the data. Instruction ID:004
- Leaderboard: Client request to be updated with the current leaderboard standings. The data entry is left blank here. Instruction ID:005
- Exit: Client requests to be logged out of their current session. Data entry is left blank here.Instruction ID: 006
- Send player choice: Utilized when a client is playing a game. The user send a command containing their column and row choice (in the data section, int, separated by spaces), along with their message to the other user if that is provided. Instruction ID: 010
- Ban user: Utilized by an admin to remove a person from the server. The data section contains the exact name of the individual who is being banned. Instruction ID: 099
- Add user: Utilized by an admin to add a person's details to the server. The data section contains the name and password for the new user (separated by a space). Instruction ID: 098

Order of message exchange:

The below sections outline the order of messages exchanged by this protocol for the main use cases involved with the project. In this application, one server interacts

with multiple clients at the same time and is able to recognize clients through established TCP connections.

User Login

To initiate user login, a TCP connection is first established. After the TCP handshaking, messages are sent in the following order.

1) The client indicates that they would like to login by sending a message including their username to the server.

2) The server looks at currently logged in users. If the user is already logged in, the received message is interpreted as the user switching devices. The result is that the server responses with information about the user's currently logged in session. This information includes: the user's status in the game room (observer or player), user's current game room number, and a list of the user's teammates if the user is a player. If the client is not currently logged in the server asks the client if it would like to be a player or an observer.

3) If client receives logged in user information, no more messages are sent for this user case and the client is considered as logged in. If the client receives the player or observer question, it sends a message to the server indicating its choice.

4) The server response to the client, confirming that it has been registered as a player or an observer.

User Logout

The user logout process is very simple in this application. The client simply sends a message to the server indicating that it would like to logout. The server then responds with an acknowledgement message. The client waits for the acknowledgment and then quits. If the client does not receive acknowledgment before timeout, it resends the logout message to the client. The client must wait for acknowledgment to ensure that the server has logged the user out. Without this, we could have a deadlock situation with the server trying to reach a client that is no longer present.

Change Client from Observer to Player or Vice Versa

This use case is simple, with only four messages being exchanged. Two messages are sent from the client to the server, and two responses are sent from the server to the client. The messages are sent in the order below:

1) The client sends a message to the server indicating that it would like to change its player/observer status.
2) The server responses asking for the new status.
3) The client sends the new status to the server.
4) The server sends a status change confirmation or error message to the client.

List the users that are logged into the system

A client is able to request to see which users are logged into the system and what game rooms they are registered in. This is achieved through the message sequence below:

1) Client sends a request for a list of users to the server. This request includes the game room that the client would like to see the list for. If the game room is not specified, this indicates a request for all users in all game rooms.
2) The server responds to the client with a list of the users that are currently logged in and registered in the requested game room.

View Leaderboard / Results

This use case is very similar to viewing a list of all users.

1) Client to Server. Please show me the leaderboard/results.

2) Server to Client. Sends the data of the leaderboard / results in string format.

Chat while playing the game (public or private messaging)

A chat functionality is available for users playing the game. Users can chat with individual clients, the clients in their game room, or all clients in all game rooms. All of this communication goes through the server, so chat messages can only be sent to clients that are currently logged into the system. The order of messages for achieving this chat functionality is as follows:

1) The client begins this process by sending a chat initiation message to the server.
2) The server responses to the client listing the possible chat options: individual, game room, entire system.
3) The client sends a message to the server, indicating its chat option choice. If the option is individual message, the client also indicates who it would like to be the recipient of the message.
4) The server then sends a message to the recipient.
5) The recipient sends a message received acknowledgment message back to the server.
6) The server sends a message to the sender indicating that the message was successfully delivered to the recipient.
7) If a timeout occurs, the sending client will begin the process again, beginning at step 1.

Ban User from the Game

To ban a user from a game room, the server admin removes the user from the game room and sends the client a message indicating that they have been banned.

Moving out of a game room, but not logging out of the system

1. Client to server. Please remove me from the current game room.
2. Server to client. Confirmation of removal from the game room.

Registering for a new game room.

Here a client picks a game room to join from a list of available game rooms. The client then tells the server which game room it would like to join. If a game room fills up during this process, the client may tell the server to add it to a game room that has since become full. The server will need to be able to handle race conditions in this concurrent system with many different clients. At the end of the process, if the server responds to the client with an error message, the client has the option to try again beginning at step.

1. Client to server. Please place me in a game room.
2. Server to client. Sends list of open game rooms.
3. Client to server. Please place me in a specific game room.
4. Server to client. Success or error message.

Game Play

Game play begins once a game room is full of clients. The server assigns the clients to their respective marks. Once each client responds with a ready message, the server displays the game board and asks the client playing with an ‘X’ for their move. Once the move is received, the server processes the move and make the client’s move for the turn. The new resulting game board is then sent to all clients, and the client that is next to play is sent a request for their move. This process repeats until the game has a winner or ends in a draw. The results of the game are then sent to each client.

The order of messages is below:

1. Server to all clients. List of players in the game.
2. All clients to the server. Ready to play message.
3. Server to all clients. Display game board.
4. Server to client. Message asking for place to put mark.
5. Client to server. Message indicating place to put mark.
6. While game has not ended, repeat from (3), switching clients each time.
7. If game has ended, display game board and results of the game.

Message handling:

The sections below describe the actions that take place on both the client and the server after certain events occur. Here all of the events refer to messages that are received by the client or server.

Server

Login Handling

When the server receives a login request with a given username, the server will check all game rooms to see if the user is currently logged in to the system. If the user is logged in, the server searches for all of the data related to that user. This information is then sent to the client.

If the user is not logged in, the server creates an instance of a user for the new client. The server then responses asking the client if it would like to be a player or an observer.

After receiving the client's answer to the player or observer question, register the user in a game room with the appropriate player/observer status.

Logout Handling

When a logout request is received from a user, remove this user from any current game rooms, and the list of all users currently registered in the system. Send a message back to the client to notify that the logout request was successful, or an error message if an error occurred.

Observer to Player Change Handling

When the appropriate data is received from the client, remove or add the user to a team and changes the user's status as appropriate. Send a success or error message back to the client.

List Users Request Handling

If a game room is specified, return a list of all users in the game room to the client. Otherwise, return the global list of all users logged into the application. If no users are found, or an error occurs, send an error message to the client.

Results / Leaderboard Request Handling

When this request is received from the client, calculate and return the current leaderboard standings to the client.

Client

Move Request Handling:

Ask user for input data for move. Send this inputted data back to the server.

Specific Game Room Request Handling:

Ask the user for input data for the game room. Respond to the server with a message indicating the game room that is chosen.

User Case Confirmation Message:

No actions required by the client here. Continue with the next step in the flow of the program.

Use Case Error Message:

Begin use case process again but beginning again from step 1 of that use case.

Player or Observer Request Handling:

The user is asked for input indicating the choice of player or observer. The inputted data is then sent to the server.

User interface:

First Screen:

```
Hello and welcome to the CPSC 441 Tic-Tac-Toe Game!

Please enter your Login ID below (Don't have one? Contact the Server Administrator to
register):

James675

Please enter your password below:

CPSC441

Hello James675. Here are a list of available game rooms for you to choose from:

Room 1: 0/2 Players

Room 2: 1/2 Players

Room 3: 0/2 Players

Type 'options' to see what you can do!

options

'join <room number>' to join a room

'List' lists all users registered on server

'create <room number>' to create a new room number
```

‘delete <room number>’ to delete a room (there must be no one in the room to do this)

‘observe <room number>’ watch a game (must be 2/2 people in a room)

‘leaderboard’ prints the current leaderboard

‘exit’ to terminate your connection

leaderboard
LeaderBoard:

RANK 1

Joe782

20 Wins; 5 Loses

#2: James675: 15 Wins; 11 Loses

#3: Mary801: 3 Wins; 1 Lose

...

Server Admin’s View:

Hello and welcome to the CPSC 441 Tic-Tac-Toe Game!

Please enter your Login ID below (Don’t have one? Contact the Server Administrator to register):

admin12

Please enter your password below:

password

Hello, admin. Here are a list of active game rooms

Room 1: 0/2 Players

Room 2: 1/2 Players

Room 3: 0/2 Players

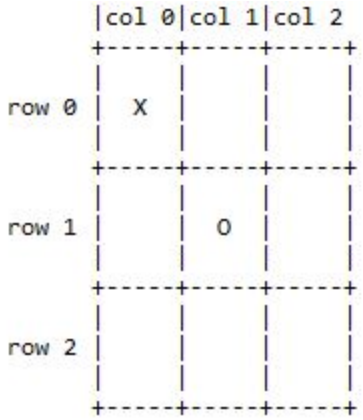
Type ‘options’ for admin privileged actions, ‘useroptions’ for all other actions.

options

‘Ban <username>’ will remove individual from server

‘Add <username> <password>’ will add individual to the server

Playing the game:



James675, it is now your turn.

Enter your column below (type options for help)

options

‘forfeit’ concede the win to Joe782

‘message <your message>’

message nice move!

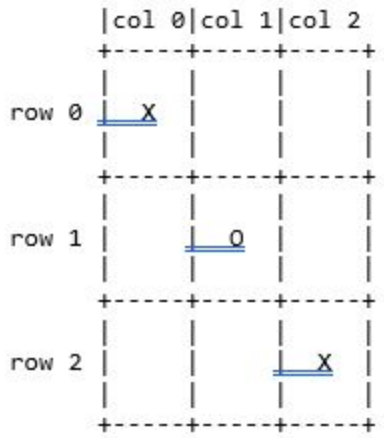
You said: “nice move!”

Joe782 said: “Thank you”

2

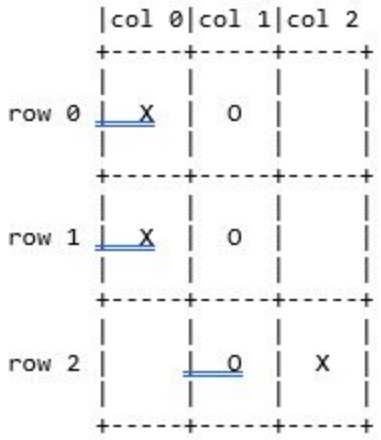
Please enter your row number (type options for help)

2



It is now Joe782’s turn. Please wait your turn (type options for help)

End of the game:

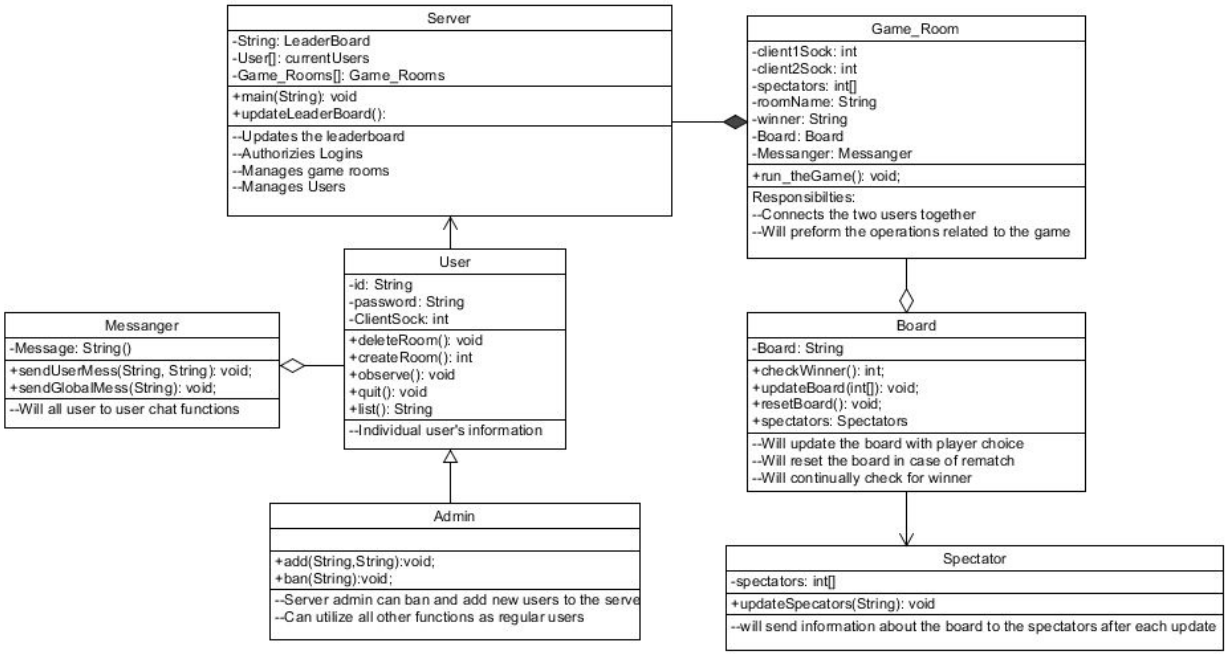


Joe782 has won the game! Joe782 is rank 1! You are rank 2!

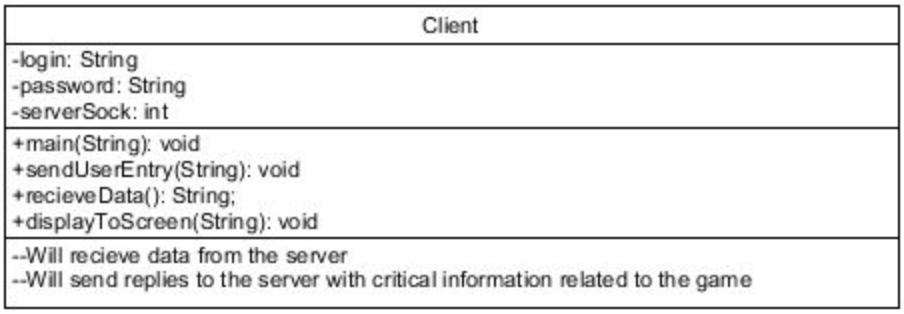
Type rematch to ask Joe782 to play again. Type quit to exit the lobby. If no action is taken within 15 seconds, both players will be kicked from the room.

Object-Oriented design of the implementation:

Server:



Client:



Expectations:

We intend to insure that the quality of the project is excellent. We will insure that before implementing any surplus features, the underlying network we create for the project is functioning properly as well as the core functions of the tic-tac-toe game we are creating.

The game we are creating is a tic-tac-toe game which will be able to function with the transfer of data via strings over a TCP connection. No large files will be transferred, and the latency associated with the transmission of our data will have no effect on the experience of the users.

To insure that all users have an accurate representation of the current state of the game, the server will act as a liaison for the two players. After each user has played their turn, the data will be sent to the server to be updated, and then passed onto the other player. The reason for this is to implement our observer feature. If another individual wants to spectate a game live, they will request information from the server and will be updated as the server is. This is how data transmission are expected to be implemented in our project.

The performance metrics we will be using to analyze our program are primarily related to quantifying the amount of lag present in each game. In other words, the latency between users is of concern as if it is too high, the game may feel unresponsive reducing the enjoyment of the players. One player will periodically ping the opposing player during the game and average the latency in their connection. This data will then be supplied back to server when the game ends to provide feedback on the average performance of all game. Should the average latency of one game be higher than an acceptable value we will set, an error message will be displayed on the server indicating the users and the game experiencing problems. The other performance aspects that may be of concern to other programs is of little concern to ours: most messages sent between client server are very small so bandwidth limitations are not an important consideration. No complex algorithms are present in our program either, so determining the Big O notation for our program (determining its processing speed) shouldn't be considered. A performance metric we will be utilizing on along with the measure of lag impact is player enjoyment. With the completion of our tic-tac-toe game, for all the reasons stated above, we believe there will be no real system caused annoyance to the players, but how well our product performs in their eyes is something we should take into account. We will invite players to leave feedback about possible new features, their impression of the UI, and general comments so we can improve our product in the future.

For a visual representation of the final design and GUI we expect to create for our project, please refer to the User Interface section of this report.

Plan:

The implementation of the project will be done in sections. The main sections will be the back-end game, server-client communication, front-end client development, and the lobby on the server side. The order of development will be as follows:

- Client server communication
    - Ensure the Server is ready and able to handle multiple TCP connections at once
- Server lobby
    - Create the game lobby with options for players to take
- Server game back end
    - Program the Tic Tac Toe game back end
- Client game front end
    - Create a client program that enables clients to communicate with the server
- Server game room
    - Create game rooms in the lobby to allow players to challenge each other and choose who to play a game against
- Basic gameplay between two clients
    - Ensure the gameplay between two clients works properly
- User accounts
    - Create user accounts on the server, with the ability for clients to login and logout
- Leaderboard
    - Create a leaderboard that tracks user accounts based on their performance in games
- Disconnect handling
    - Create the ability to handle disconnecting and reconnecting for users
- Messaging system
    - Implement a messaging system so users can send and receive messages