Group 5 - T04 **Group Project Design Document** **Fall 2019**

Bruin Kunnas, Austin Graham, Robert Karg, Cody Clark

1) Final Updated Feature List

Basic Features:

- User login/logout
- Single game room (have to be logged in)
- Display game to all players
- User input (turn based)
- First player (host) chooses a word/phrase
- Guess single letter or word/phrase
- Public chat

Additional Features:

- List available users online
- Create account (if not in system already)
- Ban users
- Private Chat

2)

a) Network System Overview

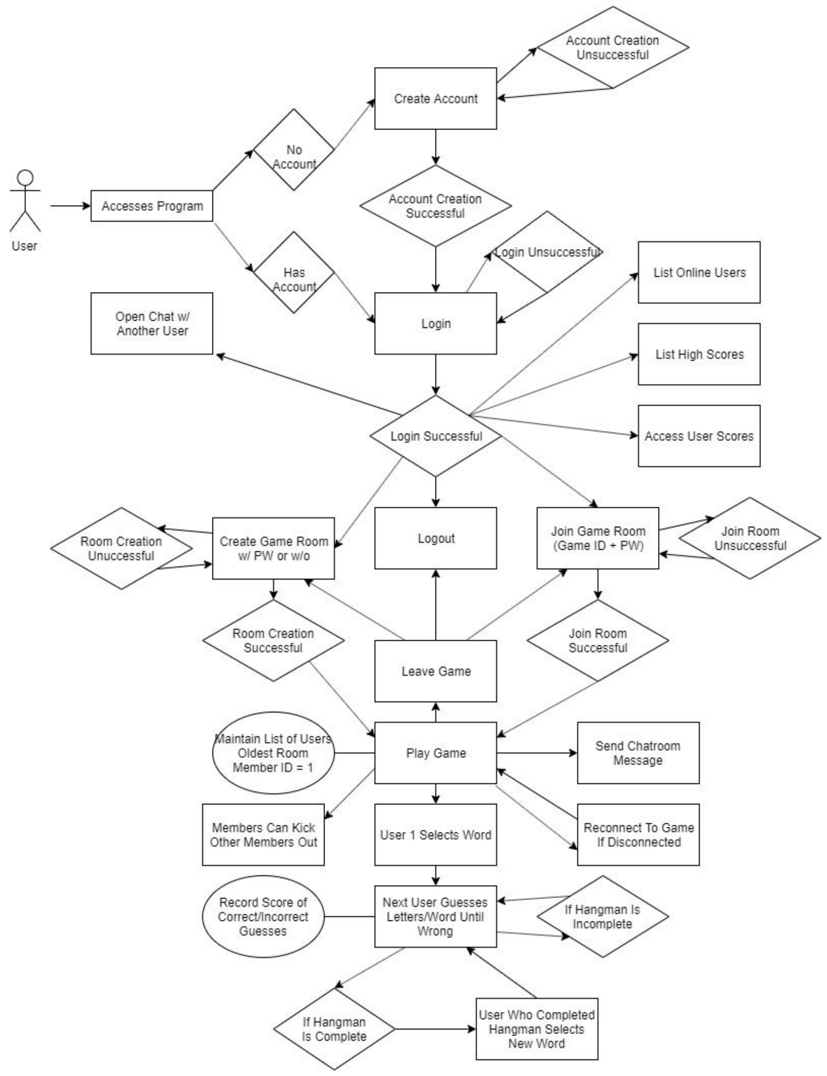
Our network system will require the client to login or create a profile for the server. Once logged in clients connect to a server where the server will send a list of commands to the client. These commands include: list users online, set word, guess letter, view previous guesses, ask for gamestate, and quit. The first player to join the server (one game room) will become the host, and the first person to pick a phrase that is going to be guessed, the other clients that join will work together in a game of hangman to guess letters or the word/phrase by taking turns between the next three players (4 total, including the host who does not play). All the players that signed into the system will join the game first, it will be limited to a maximum of 4 spots, 1 for the host and 3 for the other players.

Any other users that try to log into the system will be rejected and not allowed to log in, as the server is the game room. After a word has been guessed and the game has been completed, the second player in the list of users will be able to choose a word to start a new game, ultimately adding the original host as a player. By using the /chat command, users will be able to send a public chat message to all available users that have logged into the system. If a user exits the server, the other users will remain logged into the game and not disconnect them so they are able to continue the game. The game board will be updated after the inputs have been sent to the server, and can be accessed from any user that is logged in. All the information of the game board and the guesses will reset after the game has finished to allow for a new game to begin.

b) System Analysis

Our system uses no graphics therefore it's unnecessary to evaluate throughput as extremely little bandwidth is used.
Our system is a turn based game, therefore jitter and delay is inconsequential and will not be measured.
Instead of these we will be focusing on examining the effectiveness of the protocol, our packet overhead (Header size), and the complexity of the actions. In addition the synchronization of all of the nodes within our network is also highly important. It's necessary that all of the messages the servers and the clients send and receive are communicated in the correct order and that they do reach their destination, therefore we will be implementing a connection oriented protocol (TCP) as our underlying transport protocol.

c) Use Case Diagram


User
Accesses Program
No Account
Has Account
Create Account
Account Creation Unsuccessful
Account Creation Successful
Login
Login Unsuccessful
Open Chat w/ Another User
List Online Users
List High Scores
Access User Scores
Login Successful
Room Creation Unsuccessful
Create Game Room w/ PW or w/o
Logout
Join Game Room (Game ID + PW)
Join Room Unsuccessful
Room Creation Successful
Leave Game
Join Room Successful
Maintain List of Users Oldest Room Member ID = 1
Play Game
Send Chatroom Message
Members Can Kick Other Members Out
User 1 Selects Word
Reconnect To Game If Disconnected
Record Score of Correct/Incorrect Guesses
Next User Guesses Letters/Word Until Wrong
If Hangman Is Incomplete
If Hangman Is Complete
User Who Completed Hangman Selects New Word

Use Cases:

Upon first using the program the vast majority of commands will not be usable by the user. They start by having to either log in or to create an account. All other commands and entries will be responded to by either:

1) You cannot use that command until you have logged in. Use the following command to login, or the other command to create an account.
2) That command was not recognized. Please try again.

A user cannot enter any information unless they have logged into the system. They are able to create an account or log into an existing one, or they may enter the `/quit` command to close the program and their connection.

If the user decides to create an account they will be prompted for a username. If that username already exists in the system the server will send a message instructing them to create an account using `/signup`. Once a new username has been selected, the user will enter the username followed by their password, separating them with a space. The server will then notify the user that their account has been created and they are able to login using the command `/login` with the server first prompting them for their username followed by their password. If the username and password do not match any stored information within the database the program will inform the user that they were not successful logging in and should try again.

Once the user has logged into the system, a list of all the commands available to them will be displayed for them. If they are the first person in the game, they will also have the ability to choose the game word, but once it has been chosen, it will not be able to be overwritten until the game has finished and the next player chooses a word.

The following commands will be listed for a user after they have successfully logged into an account:

`/setWord` - Allows the first user that joined the game to choose a word, this feature will not be available to the other users that join after until after a game has been completed and the host responsibility has been reallocated.

`/gl` - Guess a letter for the game, will prompt if it was a correct/incorrect guess. All the letters will be saved into lists that can be called by any user. Will use this information to fill in the blanks of the word the user is trying to guess if the letter that was entered is apart of the game word/phrase. After the word/phrase has been correctly guessed or completed, the host ability will move to the next player that joined the game, the second player, and they will be able to choose a new game word/phrase.

`/guesses` - Shows the list of all the previous guesses for the current game.

`/gamestate` - Displays all the current information of the game including, correct/incorrect guesses, the game board, the progress of the word being guessed using *'s instead of the actual letters.

`/listUsers` - This command will list all of the currently online users.

`/logout` - At any point a user may use this command to log out of their account. They will lose their privileges and will be required once more to log in to access most of the commands.

`/quit` - At any point within a game a user can use this command in order to disconnect from the server and leave the game without affecting the other users.

`/kick` - At any point a player can kick another player from the game, which logs them out of their account so they are unable to come back into the system.

`/chat` - At any point, a user can use this command to message all users that are currently logged into the system to send them messages.

`/pm` - At any point, allows a user to send a message to one user that they have specified so that the other users online do not see the message.

When a user has successfully entered a room any messages typed and entered into the console will be regarded as chat messages, sending whatever was typed as a string to the server which the server then sends to all of the users who are currently connected to that room.

In the event that there is already 4 users currently logged into the system and another user tries to log in, the system will prompt the user that the game is full and they should try again later.

d) Feature List

Basic Features

- User login and logout
  - Allows users to use their account information to log into their account to be able to access the game, chat with other users, and guess.
- Single game room
  - Allows users to join the game which is the server only after they have successfully logged into an account in the system.
- Display the game for all users connected to the game
  - All the players will be able to view the game after the first person has picked the word/phrase for the game.
  - The board will update with the letters that have been chosen during the game on the board if they were correct or in the section below if they were incorrect. Users can ask for the gamestate at any point in the game.
- Takes user input using a turn based system to allow all users to play
  - Allow the user to choose whether to guess a letter or the word/phrase each having their own commands.
  - After one user has guessed a letter, the next player in the lobby will get the opportunity to guess a letter or the word/phrase.
  - Will continue to run through all the players until all the players exit the game room.
  - Game word/phrase will be updated according to correct letters, and the game board will be updated according to incorrect letters guessed.
- Let the first player (host) that joins to choose the word/phrase for the game:
  - The first player to join an empty game room will be the first person to choose the word/phrase.
  - This person will be passively watching the game until the word/phrase has been guessed.
  - Once the word/phrase has been guessed, the next round they will become on of the players that is able to guess letters or the word/phrase.
- Allow users to guess a single letter or allow them to guess the word
  - Have a special command so that users can choose whether to guess a single letter or guess the entire word/phrase.
  - Will compare the input of the user to the original word/phrase that will be saved in a list and compare all the chars and will return if the guess is correct or not.
  - If the user enters the command for guessing a letter but enters more than 1 letter, the system will ask the user to pick only one letter.

- ○ If the user enters the command for guessing the word/phrase but enters only one letter, the system will ask the user to enter a word/phrase.

- Public Chat:
    - ○ Allow users to chat with available users that have logged into the system.
    - ○ By not entering a special command and just typing their message, they will be able to send messages that can be seen by all users in the game.
    - ○ Use the command /chat <message> to be able to send messages to all online users.

Additional Features:

- List the available users that are currently connected to the server
    - ○ User can enter a command to list out all the users that are currently connected to the server so that they are able to choose who they message or invite to game rooms.
- Create user account (if not in system)
    - ○ Allow user to pick a username and password for their new account.
    - ○ Checks a list of all current usernames and checks to see if the username is available.

- Chat privately with one user
    - ○ To chat with a specific user using private messaging the user uses the /pm <username> <message> command.
- Ban users from a game room if they use profanity (additional feature)
    - ○ Use a special command to kick a particular user from the server
    - ○ Sends a message to the person executing the command notifying them if it worked or not, and sends a message to the person being kicked telling them they have been removed from the server.

e) Message Format: list all types of messages and their intended uses, and define format of each message type.

`/<command> <argument>`
The server will generally respond to all of the commands players may send with the result of the command and whether the command was successful or not.

Login: `/login <username> <password>`

- This command is sent to the server by the client.
- The argument provided will be the user's username.
- If the username is valid the server will then send a request for the user to enter their password.

- If the username is invalid the server will then report that this username doesn't exist, and will suggest that the user either reexamines their spelling or that they create a new account using the `/signup` command.

Logout: `/logout`

- This command is sent to the server by any currently logged in user.
- This command will log a user out and reset all of their permissions.
- This command is accessible from inside or outside a game room.
- Any room the user is currently in will be left.

Create Account: `/signup <username> <password>`

- This command is issued by the user to the server in order to create an account.
- The user will then enter a username and password of their choice.
- The server will then process whether that username already exists. If it does, the server will tell the client that the username is already taken. If the username does not exist, the server will tell the user that the creation was successful and store the supplied information in the users file.
- The server will request that the user now uses the `/login` command in order to begin playing games and interacting with other users.

List Users: `/listusers`

- The client sends the server this command.
- The server sends back a list of usernames which are currently online/logged into the game.

Quit: `/quit`

- This command is issued from the client to the server.
- This command is available to all client processes and will disconnect them from the server at any time, not affecting other users connections.

Message: `<message>`

- This is a command sent from the client to the server and distributed to all online users.
- The user must be logged in in order to use this.
- By simply typing, not using a special command, a user is able to publicly chat with users in the game with them.

Guess: `/gl <character/string>`

- This command is sent from a client in a game room to the server in order to guess a solution to the game.
- The argument for this is a guess, which may be a letter, word, or phrase.

- The server processes the request and then updates the game state, whether the guess was a correct/incorrect letter or word.
- The server will then record the results of the guess in its database, so that any user can ask for the game state at any time.
- If a client sends this message when it's not their turn, the server will let them know that it's not currently their turn

Guesses: `/guesses`
- This command is sent from a client to the server in order to guess a solution or letter for the game.
- Lists all the guesses that have been made during the current game.
- Will be cleared along with the game state when the game has been completed.

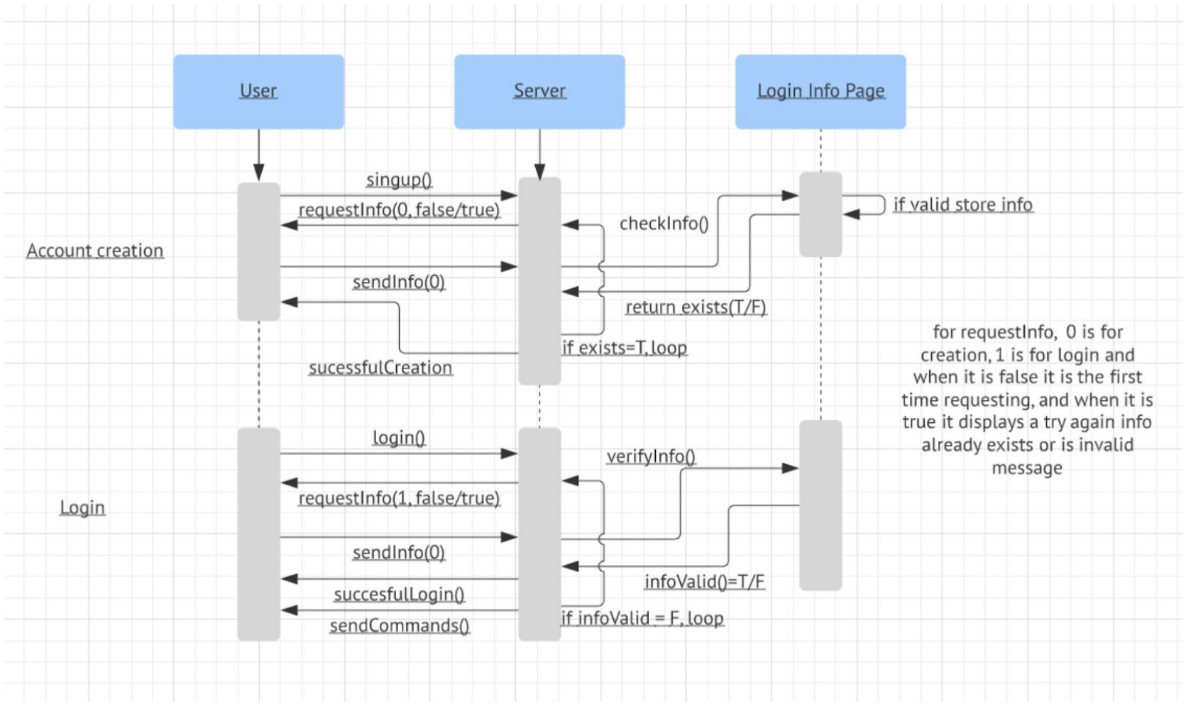
Gamestate: `/gamestate`
- This command is sent from a client to the server to get all current information of the game.
- Lists all correct/incorrect guesses and the game board.
- Shows how many guesses the players have made in total.

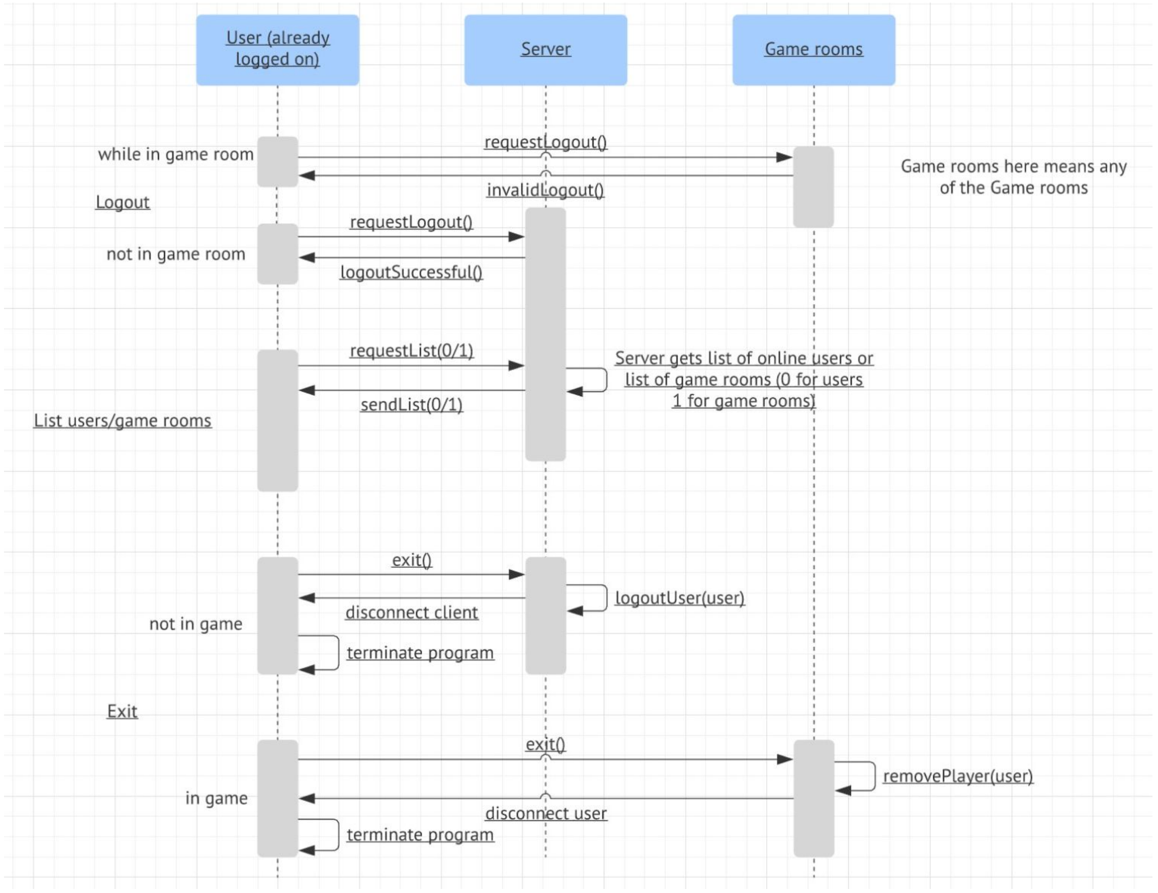
Set Word: `/setWord`
- This command is sent from the first user (host) in the game to the server.
- This allows the first player to choose the word/phrase for the game.
- Other users that have logged in after the host will not be able to use this command to prevent the word being overwritten.

Public Chat: `/chat <message>`
- This command allows any user to chat with all available users that have logged into the system.
- If a user is not logged into the system, their message will not go through to the other users.
- When the message is sent to the other users, it will specify what user sent the message, `<username>:<message>`

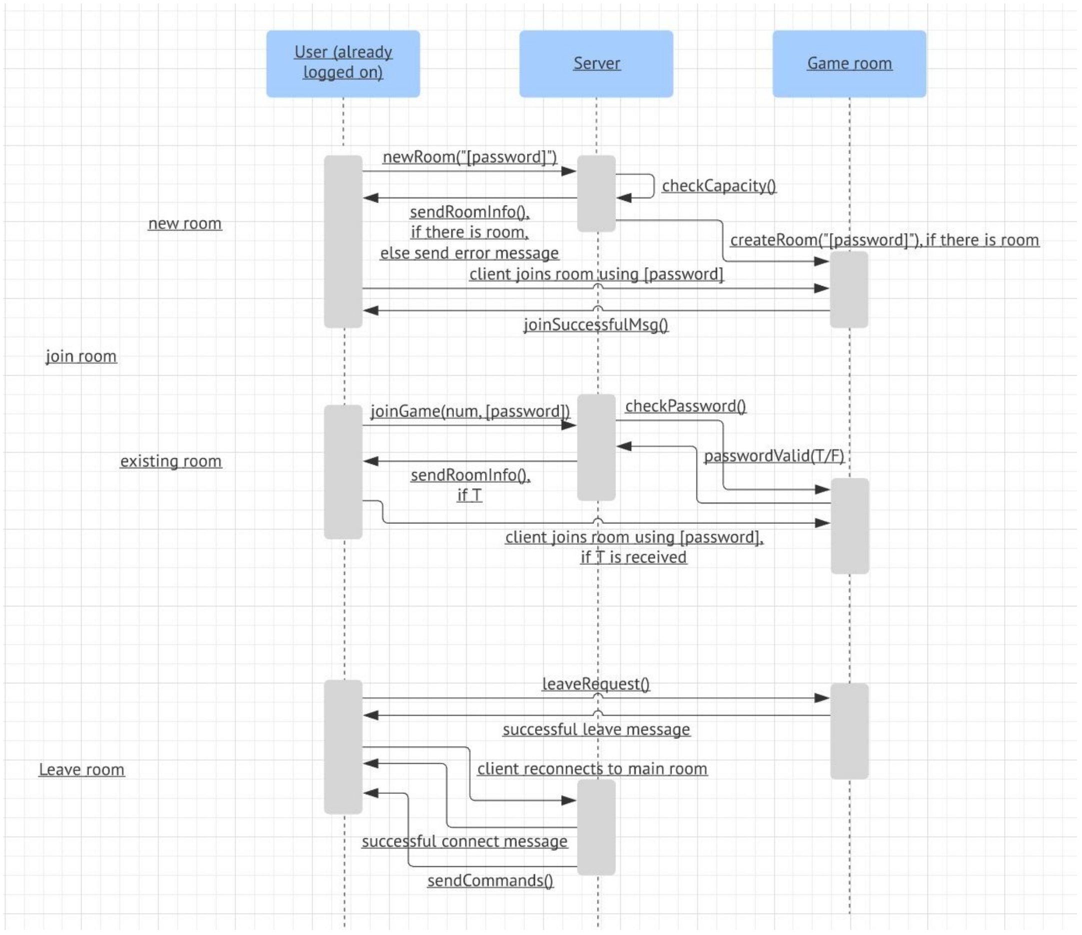
Private Chat: `/pm <username> <message>`
- This command allows any user to chat with a particular user they have specified, only if they have logged into the system.
- It will show that this is a private message for the user receiving it so that they know to not send a message back to everyone.
- `PM: <username>: <message>`

f) Order of message exchange for each use case, illustrate the message exchange in a sequence diagram.

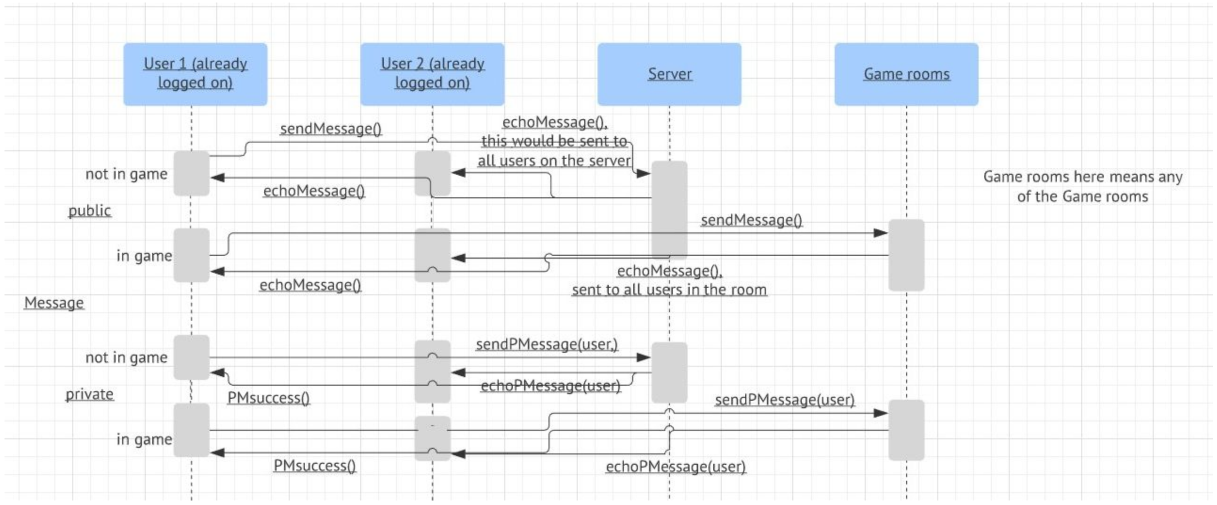


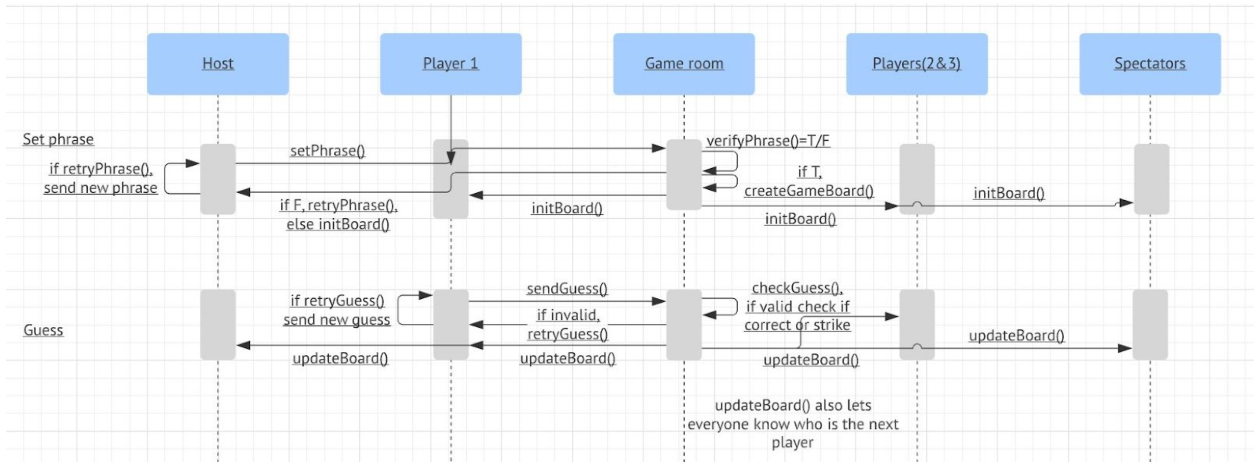


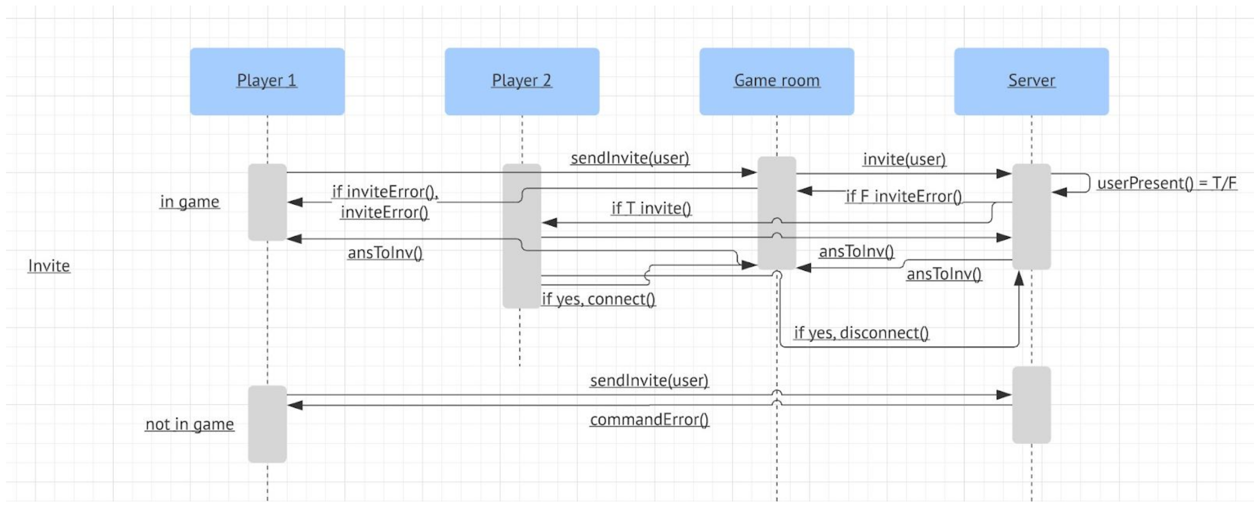
Bruin Kunnas, Austin Graham, Robert Karg, Cody Clark


User (already logged on)
Server
Game room
newRoom("[password]")
checkCapacity()
sendRoomInfo(), if there is room, else send error message
new room
createRoom("[password]"), if there is room
client joins room using [password]
joinSuccessfulMsg()
join room
joinGame(num, [password])
checkPassword()
passwordValid(T/F)
existing room
sendRoomInfo(), if T
client joins room using [password], if T is received
leaveRequest()
successful leave message
Leave room
client reconnects to main room
successful connect message
sendCommands()

Note: [password] can be nothing, in which case “” would be used. Also, leave room takes place when a user is already connected to a room







Player 1 could also be a host.

g) Message handling:

<u>Login</u> - Client Entity sends a login request message (/login).
- Server acknowledges the request and sends a request to the client for the user info
- Client sends a username and a password
- Server verifies info with the login info page.
  Responds either "incorrect user credentials"
  Or
  Responds "login successful"
- Server then sends a message to the client with a list of possible commands.

Logout - Client sends a logout request message
Logs the user out without disconnecting the user

Quit - Client sends an quit request message
Server checks if logged in(not in a game), in a game or not logged in
If not logged in the server disconnects the client
If logged in but not in a game the server logs the user out then disconnects the client

Create Account - Client sends request to create new account
- Server sends back a message “enter new username and password, separated by a space”
- Client sends back a massage “[username] [password]”
- Server checks name against existing users and responds
  Either “name not available”
  Or
  “Account creation successful, you may now login”
- username and password are stored in the login info page
- From here the client needs to login normally. They are not then automatically logged in

List Users - Client sends a request to view all users currently online
- Server gets a list of users online and sends them over to the client
- Client receives a string of users and they are printed out

Guess - Client sends input string to server
- Server checks if the input has been guessed in the past
  Either “yes”, and prompts the user for new input
  Or
  “No”, and adds input to list of guesses and updates the gamestate.
- Server changes the player that is up next, and allows them now to choose a letter/word.

Gamestate - Client sends a request to the server to view all the current information about the game.
- Server gets all information needed to display to the user that has requested the information and sends it back to the client.

Set Word - Client sends information to the server, selecting a new word to use for the game.

- Server receives the input and converts it to *'s for the game and saves the word elsewhere.
- Compares to this word every time a user guesses a letter/word to ensure their guess is correct or not.

Message - Client inputs a string with a special command to send a message either public or private.
- The server will distribute the message to all of the users that are currently logged into the system and display it if it is a public.
- The server will distribute the message to one user if it has been specified to which user it should be sent to, for private only.

Kick - Client specifies the user that should be removed from the server.
- The server then logs that user out of their account, notifying both users that they have been removed.

h) User Interface:

When a user first joins the server it will list the commands that users can use (commands for host will be shown for all users but will not work for all users) and users will use commands to be able to see the gamestate or list of users online, etc..

The game interface would be represented by a simple ASCII art stick man.
It would start as a blank interface between a series of dotted lines as such:

```
----------



----------
```

Which would then grow as the game progressed and people guessed incorrect answers.
The following represents 3 incorrect guesses:

```
----------
  O
 / |

----------
```

(Continued on next page)

Until eventually the entire body has been completed, as follows:

```
----------
  O
 /|\
 / \
----------
```

Under each game state would also display current statistics for the game:

```
----------
  O
  |

----------
Phrase: *ei*e
Current Guesses: d, e, i, b
Current user guessing: <username>
Next player guessing: <next_username>
----------
```

Any additional GUI will essentially be the server responding to requests by the user.

All messages will simply be sent to the server as single line requests, with the server parsing the number of arguments passed after the request. So for example:

```
/login user1 1
```

Successful login

To which the server would send back:

Sever [8:52:46]: enter username and password (separated by a space)

The user would then enter:

```
/login user2 2
```

Then the server would respond with:

Successful login

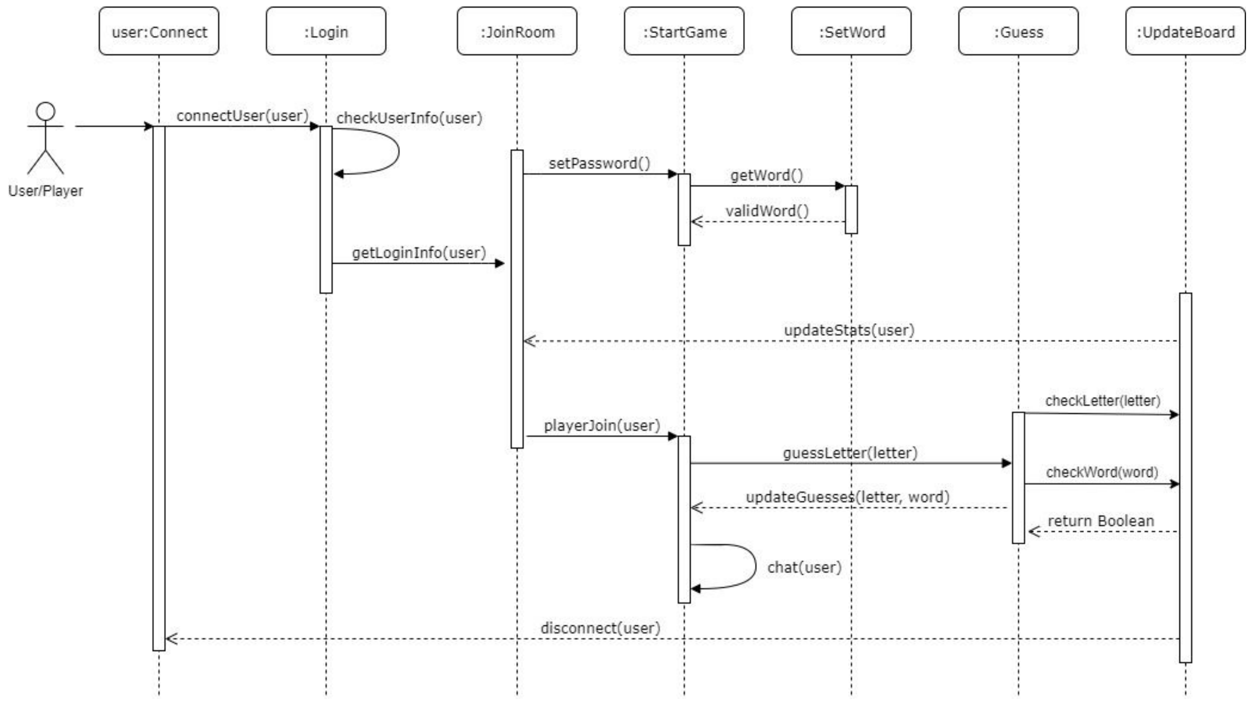
Or:

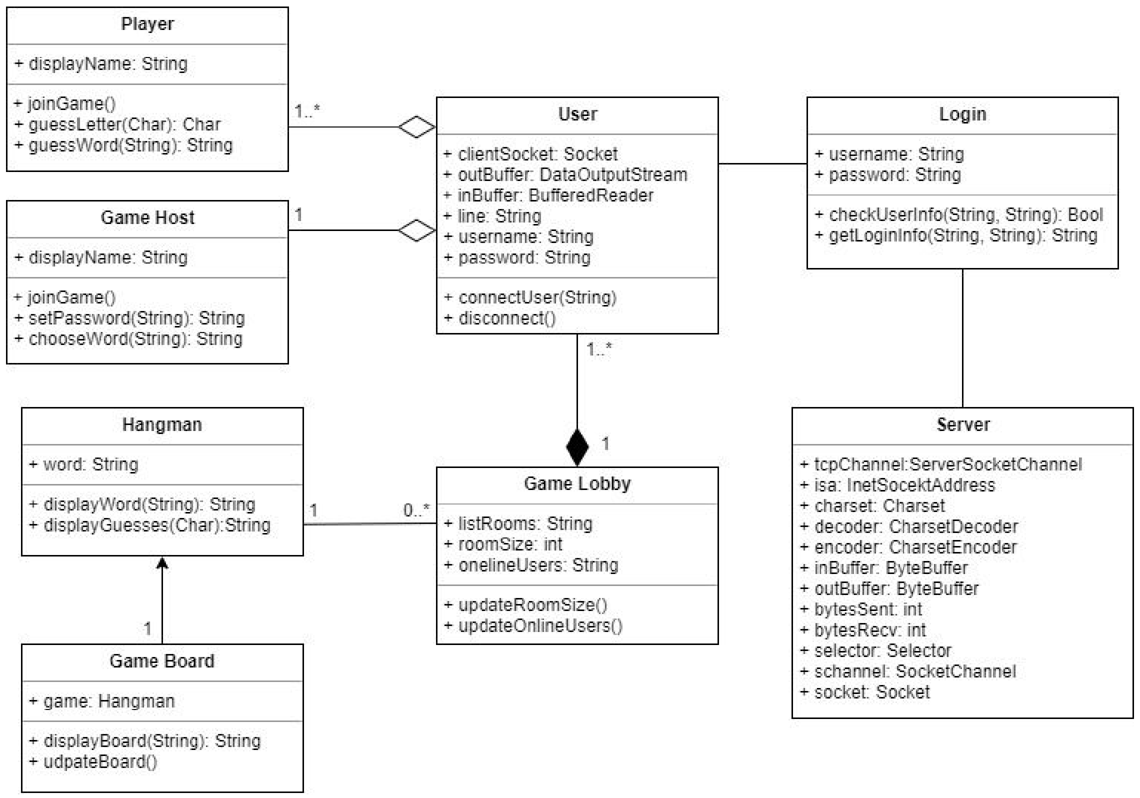
Unsuccessful login

For simplicity's sake the password as entered on the client side will be plaintext. Please don't use your actual passwords when playing our game. We cannot guarantee security. Game room messages will be formatted as follows:

```
pm: Username: Example message
```

i) Object-Oriented design of the implementation:





The diagrams above help illustrate the implementation of our object-oriented design. These will not reflect the final design of our project but they will help guide us in the direction we wish to go. We will have our server be the main factor of our design. It will create the lobby for the game along with checking all the users information when they are logging into the system. Each user will be given different functions depending on the number of players currently logged into the system. If a room is empty, the user will be given the role of game host where they will be able to set the password for the room, if they want it private, and choose the first word/phrase that will be used for the game. Any other user that enters after the first one, up to 4 users total including the host, they will become players and will be given the ability to guess letters and the word/phrase.

There will only be one game board (hangman) and after each game has been completed, the board will reset, give the next player that joined the room (second overall) the ability to choose a word/phrase (game host) and the previous game host now becomes a player. It will rotate through all the players as many times as they choose. It will continue until all players have left the room. Each player and game host will have the ability to chat while in the game with the other users in that room. Each user is also able to chat privately with any user online using the special commands listed previously. A public message will check for all the online users and send that message to them, making sure that all users have been logged in. If it is to a particular user, it will check that the user is online (logged in) and then the message will be sent. A user is able to quit or logout at anytime, the game will continue with the remaining players in the game, quit will completely close their connection to the server, while logout just removes them from the system so they would have to login again.

j) Evaluation

To address the various scenarios presented in the system analyses we will now go through each individually.

Firstly the effectiveness of the protocol, packet overhead, and the complexity of the actions.

The protocol that we use is a simple call and response between the client and the server. The client sends a message to the server, which the server processes and then sends the result back in the form of a string to the client simply to inform them of the result, which is then printed to the client's terminal. This is as simple as a messaging protocol which reports results back to the user can be. The operations at the server level also do not depend on the response sent back to the client, the client doesn't perform any operations with it other than to display it to the console. Without sending a response back to the user the server would still be able to create and perform game actions, making this system robust. (However it would mean it'd be pretty unsatisfying to play as a user)

The packet overhead is a compromise on usability in the programmers favour, which we decided was appropriate due to our game not being time sensitive, and because the overhead is still relatively small. (It's not as if we're regularly transferring files filled with MBs worth of additional data. The user enters a command of the form <command> <possible argument1> <possible argument2> which is then sent to the server to be processed. Commands are always at least 3 characters long, and the arguments may be as long as the possible Java String object length. This could be shortened into a smaller packet by boiling the commands down to small numbers (1-13) by converting them on the client side before they're sent to the server side, however that introduces more calculations on the server side and reduces readability for the programmers. The arguments however are in as small of a form as they can be, as it's difficult to convert a password into something smaller than whatever that password's string length may be. Likewise the server could send single or double digit replies to the client, which the client would then need to interpret to produce the correct response to print to the user's console, but once more that opens up room for error prone dependencies if code on either side of the server/client were to change, as well as introducing more processing user for the client. Because the messages exchanged between the server and client are simply strings, we estimate that the additional header/packet size to be less than a KB for any message sent.

The complexity of actions taken on the client side is extremely low. The client simply needs to enter input through the console which is then sent to the server for all of the processing. Then the server issues a reply to the client which the client then displays to the terminal. The client side uses multithreading inorder to achieve concurrent input and output. A thread is dedicated to receiving input from the client which it then sends to the server, and a seperate thread is dedicated to receiving output from the server and displaying it.

The complexity of actions taken on the server side are slightly more complex, but essentially boil down into a series of if/if else/then statements. The server receives the commands from the user and then examines the beginning of the message. (All messages must start with a command) The if/then sections are dedicated to parsing the information and using it in an appropriate manner. Most of the server actions are inaccessible unless the user has logged on. Any command or input that isn't recognized will be responded to simply with an "invalid command" response and a prompt for the user to try again. This workflow allows the programmer to add additional commands with ease by simply opening up another if/then statement that looks for that particular command. Issues would come up if one command had the same initial letters as the entirety of a different command (i.e. if we had implemented the commands /usersOffline as well as /users) which would then be evaluated depending on the first if/then statement reached that would be valid, however our project does not currently possess any such issues. This issue would also be mitigated with the additional design requirement to place specific and longer command evaluations before smaller and simpler evaluations. (i.e. Evaluate commands that begin with /usersOffline before evaluating commands that simply begin with /users)

Finally we will examine the synchronicity of the nodes within our network. Because our game can take place in a slower manner the timing of clients sending and receiving messages is important but non-critical. The server runs a SelectServer operation when determining which sockets to read data from, which it then processes in the order in which that SelectServer operation selected the servers. If client A was selected first followed by client B, client A's input would be processed completely before the server would then move onto processing client B's input. In order to ensure turn appropriate actions within the game are taken, we have specified on the server side a host which has special permissions and a turn order that's iterated through every time someone guesses an answer incorrectly. If the host would logout/disconnect then the server would assign host privileges to the next player in turn order. Should a user try to guess a letter out of turn order, the server will inform them that it's not their turn. As soon as it becomes their turn, the server will then inform that player that they are now able to guess a letter.

On the client side there is no processing of the game state, only messages being sent to the server and from the server, so that synchronicity is unimportant. Due to the server implementing SelectServer, it's guaranteed that the user's input and the user's output will be generated within the same block of server processing. The server must respond to the client after receiving input before it begins processing another client's input. This way it's ensured that the client receives the correct output from the server.

TCP was used to ensure that these client messages do reach the server, and the server's message do reach the client. If a user has entered a command to be sent to the server then it's guaranteed that the server will receive it as long as all of the layers underneath the network layer are functioning correctly. (Essentially as long as the client and server both have an internet connection)

The one evaluation we could strive to improve upon is error checking. For select commands (/setWord & /gl & /kick) error checking has been included and is operational, however this could be expanded to additional commands such as /pm and /signup as well as be made an emphasis for the entirety of the code base. The system is highly usable, however additional tests to ensure that the system would still function in case of user error would lead to a healthier, more stable system.

Through this exploration and evaluation of our system we believe that we have considered and implemented a successful and usable system that adheres to and attempts to preserve the high quality of service necessary to play the game, while also acknowledging some concessions that were made in order to create a simpler codebase which reduces the chances of introducing unwanted errors into the system. The one area where improvement could be made is to simplify and standardize error checking as already mentioned.