

CSE310 Final Project  
Group # 43  
Members Cho Ki Bbum, Fieger Brandon, Yin Jonathan

## OVERVIEW DOCUMENTATION

### For TIC-TAC-TOC Multilayer Game

The entire project has been implemented including the bonus tasks. Playing games, listing current players and ongoing games, observing games and commenting have been completed.

## USER DOCUMENTATION

### How to play:

1. Open a terminal and run

```
python server.py
```

2. Open another terminal and run

```
python client.py localhost 50000
```

3. In the test client terminal, enter a username. Then you enter any of the following

```
WHO  
GAMES  
PLAY {username}  
EXIT
```

4. Open another terminal and run

```
python client.py localhost 50000
```

5. In the new terminal enter a username. Then you can start a game with the other player

```
PLAY {first username}
```

*Note: You can only play another player if they are not busy (they aren't already in a game)*

6. Then you can bounce back and forth between terminals and entering

```
PLACE {num}
```

- 7a) To observe a game, you can enter

```
OBSERVE {num}
```

Where num is the id of the game to observe. Game ids can be found using the GAMES command.

*Note you can only observe 1 game at a time and you must unobserve before observing a new game.*

b) To stop observing a game, you enter

UNOBSERVE {num}

Where num is the id of the game you are observing.

8. You can comment to other people in the server using

COMMENT {message}

You must be playing or observing a game to comment.

## SYSTEM DOCUMENTATION

The game server works from communication between three main bodies, the Client, the Server, and a Game instance. First, in order to launch a game, a server is created using python server.py. The server will listen to incoming connections by utilizing I/O multiplexing (this is how concurrency is implemented) to listen to multiple different clients. The server also has a GameServer object whose goal is to keep track of the status of players and games that are currently active. When a client socket logs in to the server, they are presented with a login screen, after entering a unique name (which is checked by the GameServer), the GameServer adds a Player object to a dict from sockets to Players which describes the status of the client(busy or available), the client's username, as well as the game the player is in. It's notable that the actually game logic is handled in the TicTacTocBoard object, not the GameServer. From now on, clients will be able to send messages to the server. In order to simplify and streamline processing input from Clients, messages sent by the client are wrapped in a ServerProtocol object which checks if the command typed by the client is valid as well as if the number of arguments is also valid. If the ServerProtocol detects no errors, then the command is then processed by the server. Depending on the command the client gives, the following actions will take place

### **FOR ALL COMMANDS, IF THE USER IS NOT LOGGED IN, SEND STATUS CODE 401.**

Help: The server will send a static string with descriptions on the command types and their usage with status code 212.

Who: The server will query the GameServer on all of the players logged into the server, create a string representation of them (their usernames), and send them back to the client with status code 251.

Games: The server will query the GameServer on all of the games that currently are being played, create a string representation of them, and send them back to the client with status

code 250.

Place: The server will get the game the client is currently playing by getting its corresponding player object. It will then parse its second argument in order to extract the position to place a new mark at for the player. It responds with success using status code 200. After it does so, both players are informed the move that is made using status code 201 and make the corresponding move on their own board (each client has their own board object which they make moves on). If the game is finished, both players are informed utilizing status code 202 for if a winner is found. Otherwise, status code 203 is used for a tie. Cleanup then occurs (the references to the board are erased by the GameServer and the Players and the status of the

Error possibility 1: If the player is not playing a game, send status code 400

Error possibility 2: If the player is not in the game being referred to, send status code 414

Error possibility 3: If the player makes an illegal move, send status code 411

Error possibility 4: If it's not currently the client's turn, send status code 406

Error possibility 5: If the client creates a poorly formatted move, send status code 424

Exit: Server removes socket to player mapping and sends status code 206 to the client, informing the client to close its socket. If the client was playing a game in the meantime, status code 205 is sent to the opposite player as well as all observers to inform them that the other player has left.

Play: GameServer queries for the Player Object with the username provided by the client, if successful, status code 200 will be sent to both players. After that, a new TicTacTocBoard object is made and tracked by the GameServer. The server sends status code 210 which will prompt each player also makes their own TicTacTocBoard object.

Error Possibility 1: If the client attempts to play with a non-existent opponent, then status code 404 is sent to the client.

Error Possibility 2: If the client attempts to play with itself, then status code 409 is sent to the client.

Error Possibility 3: If the client attempts to play with a busy player, then status code 423 is sent to the client.

Observe: Server queries GameServer for the TicTacTocBoard with the same I.D. as the player specified. If successful, the Player object associated with the client is changed to status code 0, which represents an observer. The board also adds the player to a list of observers and a link to the game is made by the player through an object field. Status code 220 is sent to the client which will detail how to create a representation of the current game on the client side. Every time a player places a valid move, the server will send Status code 220 to all of its observers.

Error Possibility 1: If the client is currently busy, send status code 408.

Error Possibility 2: If the client's game is non-existent, send error code 410

Unobserve: The GameServer queries the Player the client is associated with for the game that it is observing, it then disassociates that game from the player as well as removes that player from the GameServer's list of observers. This will prevent any future updates to this client on this game.

Error Possibility 1: If the client is not an observer, send status code 421

Error Possibility 2: If the client is not observing the game with the I.D. they provide, send status code 430

Comment: The GameServer queries the Player the client is associated with for the game the player is currently in. It then sends the message to all observers and players in the games with status code 230.

Error Possibility 1: If the player is not in a game, send status code 425.

Additionally, each status code is additionally either supplemented with a message or not. Nevertheless, the client actually has two different processes. One process is solely dedicated to parsing and sending User Input while the other process solely focuses on handling server responses. When the server responds with a message, the message is first wrapped in a ClientProtocol object which performs a similar function to the ServerProtocol object, it checks for errors in the server response. If there doesn't exist errors, then the Client will parse out the appropriate status code and print out an appropriate response based on error code.

In order to add functionality, the following must be done: An additional command must be defined in the ServerProtocol so that it isn't detected as an error, the server must handle the message in some way as well as respond to the client with a message with a status code. The new status code must be added to the ClientProtocol so that it isn't detected as an error. Finally, the testClient must handle the message sent by the server in some way.

## TESTING DOCUMENTATION

### Testing scenarios

- **Logging in**

*Case:* IF username name is taken

*Expected Output:* Server responds with an error

*Actual Output:* Server responds with an error

*Case:* IF successful

*Expected Output:* Server returns with success

*Actual Output:* Server returns with success

- **Displaying all games**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Server returns with list of games

*Actual Output:* Server returns with list of games

- **Displaying all players**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Server returns with a list of players

*Actual Output:* Server returns with a list of players

- **Starting a new game with a player**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF already in a game

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF player not found

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF player is busy

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF player is same as logged in one

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Starts a new game with player

*Actual Output:* Starts a new game with player

- **Placing a new move for a game**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF not in a game

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF ill-formatted move

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF move is illegal

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF not turn

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Place the move onto the game, notify if won/draw

*Actual Output:* Place the move onto the game, notify if won/draw

- **Leaving a game**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Terminate any ongoing games, close program

*Actual Output:* Terminate any ongoing games, close program

- **Help message**

*Case:* Display help message

*Expected Output:* Show help screen

*Actual Output:* Show help screen

- **Observe a game**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF in a game already

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF game does not exist

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Begin observing a game

*Actual Output:* Begin observing a game

*Case:* New move from observed game

*Expected Output:* Move is updated on observer game board

*Actual Output:* Move is updated on observer game board

*Case:* Observed game ends/wins/draws

*Expected Output:* Game board is updated/client is notified

*Actual Output:* Game board is updated/client is notified

- **Un-observing a game**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF not observing a game

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF game does not exist

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Game no longer is updated/observed

*Actual Output:* Game no longer is updated/observed

- **Commenting on a game**

*Case:* IF not logged in

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF not observing a game

*Expected Output:* Server returns an error

*Actual Output:* Server returns an error

*Case:* IF successful

*Expected Output:* Comment is sent to all participants in game

*Actual Output:* Comment is sent to all participants in game