

Names: Jake Tusa, Michael Cheung, Billy Ko

Group 40

Date: 5/8/17

CSE 310 Final Project Server Documentation

TASKS COMPLETED: Parts 1 and 2

=====USER DOCUMENTATION=====

OBJECTIVE OF REVERSE TIC-TAC-TOE

The game has two players and involves a 3x3 board. One player will use 'X' pieces, while the other uses 'O' pieces. The two players will alternate and place their pieces onto the board one at a time. The first player to place three pieces in a straight line loses the match. If there are no more available spots on the board but neither player has three pieces in a straight line, then the match is a draw.

SETTING UP A SERVER

INSTRUCTIONS

PART 1 SERVER

1. Run serverp1.py in a command line

PART 2 SERVER

1. Run serverp2.py in a command line

RUNNING THE CLIENT

COMMANDS

help	this command takes no argument. It prints a list of supported commands, which are ones in this list. For each command, it prints a brief description of the command function and the syntax of usage.
login	this command takes one argument, your name. A player name is a userid that uniquely identifies a player. Your name is entered with this command and is sent to the server.

place	this command issues a move. It takes one argument n, which is between 1 and 9 inclusive. It identifies a cell that the player chooses to occupy at this move.
exit	the player exits the server. It takes no argument.
games	this command triggers a query sent to the server. A list of current ongoing games is returned. For each game, the game ID and game players are listed.
who	this command has no argument. It triggers a query message that is sent to the server; a list of players who are currently logged-in and available to play is retrieved and displayed.
play	this command takes one argument, the name of a player X you'd like to play a game with.

INSTRUCTIONS

PART 1 CLIENT

1. Run clientp1.py in a command line with two additional arguments:

1. The name of the machine running the server
2. The port number that the server is listening at

If the machine you are trying to connect to is not running the server or does not exist, or if the port number is incorrect, then you will receive an error message with the reason for the error.

2. After connecting to the client, use the “login” command followed by a username. If the username is not already taken, then it will become your username. Otherwise, the server will return an error message stating that the username is unavailable and the user must use the login command again. Using the “place” command at this stage will return an error message and nothing else will happen.

3. Once you have logged in, the client will then wait until the server has two players to begin a new match of reversed tic-tac-toe. Once the server finds two players, a new match of reverse tic-tac-toe will automatically

begin, and your client will display the tic-tac-toe board as well as the opponent's username.

At any time, you may use the command 'exit' to exit the server and log out, ending any active matches.

4. You will be notified via command line when it is your turn. During your turn, use the "place" command followed by the position on the board that you wish to place your piece on. If the position is already occupied, or the number you entered is invalid, you will receive an error message and must use the "place" command again to make your move. Using the "place" command when it is not your turn will return an error and nothing else will happen. After successfully making a move, the client will display the updated tic-tac-toe board and you must wait for the other player to make his/her move.

5. Once one player has 3 pieces in a row or if the board is fully occupied, the client will display whether you won, lost, or drew. You will then automatically wait for another match to begin.

6. To stop playing, use the "exit" command.

PART 2 CLIENT

1. Run clientp2.py in a command line with two additional arguments:

1. The name of the machine running the server
2. The port number that the server is listening at

If the machine you are trying to connect to is not running the server or does not exist, or if the port number is incorrect, then you will receive an error message with the reason for the error.

2. After connecting to the client, use the "login" command followed by a username. If the username is not already taken, then it will become your username. Otherwise, the server will return an error message stating that the username is unavailable and the user must use the login command again. Using the "place" command at this stage will return an error message and nothing else will happen.

3. Once you have logged in, the client will enter a lobby and you must use the “play” command to start a game with another player or wait for another player to use the “play” command to start a game with you. At any time, you may use the command ‘exit’ to exit the server and log out, ending any active matches.
4. You will be notified via command line when it is your turn. During your turn, use the “place” command followed by the position on the board that you wish to place your piece on. If the position is already occupied, or the number you entered is invalid, you will receive an error message and must use the “place” command again to make your move. Using the “place” command when it is not your turn will return an error and nothing else will happen. After successfully making a move, the client will display the updated tic-tac-toe board and you must wait for the other player to make his/her move.
5. Once one player has 3 pieces in a row or if the board is fully occupied, the client will display whether you won, lost, or drew. You will then automatically enter the lobby until you or another player uses the “play” command to put you into a match.
6. To stop playing, use the “exit” command.
7. At any given time, you can use the “games” command to see the IDs of ongoing games and the names of their players, or the “who” command to see the player names of all logged in players.

GAMES POSSIBLE

Theoretically, there is no cap to the number of games that the server can run concurrently. It continuously accepts connections from clients and will always create a new game when a “play” command is successful.

=====SYSTEM DOCUMENTATION=====

REVERSE TIC-TAC-TOE PROTOCOL

METHODS:

210 LOGIN

The LOGIN method means an attempt to for a client to log into the server. The contents of the request includes an ID set by the user.

211 PLACE

The PLACE method means an attempt for a client to place a piece on the tick-tac-toe board. The contents of the request includes the position on the board in which the player wishes to place a piece on.

212 EXIT

The EXIT method means that a client has exited from the game.

222 WHO

The WHO method means a request for the IDs of all available players connected to the server who are not in game.

223 GAMES

The GAMES method means a request for all active game IDs and the IDs of its players.

224 PLAY

The PLAY method means a request to play with a specific player. The contents of the request includes the ID of the target player.

STATUS CODES:

200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request, for example:

PLACE the status of the newly updated tick-tac-toe board

213 WAIT

The WAIT status code is used for two scenarios:

1. The login request is successful, but there are not enough players to begin a match. The client must wait for a match to begin.
2. The client is participating in a match, but it is the other player's turn to make a move. The client must wait for the other client to make a move before it can make one.

214 START

There are enough players to begin a match, and a match has begun.

215 GO

It is the current player's turn in the match

216 WON

The match has ended and the client has won

217 LOST

The match has ended and the client has lost

218 TIED

The match has ended and both clients tied

219 NAME

Contains the name of the opponent player

220 LEFT

The opponent player has left the match, and the match has ended.

221 DISPLAY

The game board has been updated. Contains the status of the game board.

400 ERROR

An invalid request was received or an error occurred executing a request.

REVERSE TIC-TAC-TOE SERVER

SERVERP1.PY

*** This is the server for Part 1 of the final project

LANGUAGE

Python 3.5

MODULES

socketserver	used to simplify the task of writing network servers by server objects with handlers and stored client information
threading	used to create multiple threads to multiple clients simultaneously
time	used for the sleep method which pauses a thread

GLOBAL VARIABLES

The server class contains global variables for each of the protocol methods and status codes, which can be found in section 1.1 and 1.2 in this documentation, as well as local variables:

connections	a counter of the number of connected clients
playerList	a list of player objects representing the current players
nameList	a list of IDs of current players; used to check if an ID is available for login
game	a Game object to represent status of the match

CLASSES

ThreadedTCPHandler	a handler object which contains a handle() function that does all the work required to service a request
ThreadedTCPServer	a server object that uses a handler object to handle all requests to the server
Player	an object to represent each player
Game	an object to represent an ongoing match

ThreadedTCPHandler(socketserver.BaseRequestHandler)

FUNCTIONS

handle(self)	handles all requests from the client to the server
---------------------	--

ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer)

Inherits the functions and variables from both socketserver.ThreadingMixIn and socketserver.TCPServer

Player

VARIABLES

name	name of player
state	represents whether the player is busy or available
piece	shape of the player's piece, 'X' or 'O'
isTurn	flag to indicate if it is the player's turn
playerWaiting	flag to indicate when a player is waiting for his/her turn
playerExited	flag to indicate if a player has left the match

METHODS

init	class constructor that automatically runs when instantiated
getName	returns name
getState	returns state
getPiece	returns piece
getIsTurn	returns isTurn
setName	sets name
setState	sets state
setPiece	sets piece
setIsTurn	sets isTurn

Game

VARIABLES

NUM_PLACES	static variable for the number of positions on the board
BLANK	symbol for an available position on the board

TIE	return value when a game ties
playerList	list of players in the match
gameBoard	list with each index representing a position on the board
isActive	flag to indicate that the match is active

METHODS

getPlayerList	returns playerList
getIsActive	returns isActive
setIsActive	sets isActive
addPlayer	adds a player to PlayerList
removePlayer	removes a player from PlayList
createBoard	initiates gameboard
displayBoard	returns a string visualization of the current game board
updateBoard	updates the game board with a new piece
checkLoser	checks board for losing or tie condition

CODE

handle(self)

Variables

The function references all the global variables that need to be shared.

“killThread” is a variable used for control flow that is checked when a player exits

Sleeping threads

Before each time the server sends a message to the client, it calls “sleep” to pause the thread to avoid race conditions or sending too many messages too clients at once.

Accepting incoming connections

If there are less than 2 connections, then upon accepting a connection, the server prints a message and sends an “OK” protocol message to the connected client.

If there are already 2 connections, then the connecting client will receive an “ERROR” message.

Handling commands or messages before a game has begun

A variable “`loginSuccess = False`” is used as a flag to indicate if a player has logged into the server and is set `True` when a player successfully logs in.

A while loop is executed while `loginSuccess` is false, in which the server receives messages from the client.

If the message is a “LOGIN” request, then the server checks if the name is available and returns “OK” to the client if it is, and “ERROR” if it isn’t.

If the message is an “EXIT” message, then the server returns an “OK” message and sets “`killThread = True`” and exits the function.

For all other requests or messages, the server returns an “ERROR” message to the client.

After a player logs in, if there are no players in the `playerList`, then a player object is created with “`player = Player(name, "available", "X", True)`”, and assigning the ‘X’ symbol to that player. If there is already a player in the `playerList`, then a player object is created and the ‘O’ symbol is assigned to the player.

If there is only one player connected, a “WAIT” message is sent to the player.

Starting a game

Once a second player is connected, a game object is instantiated and the two players are added to the game’s `playerList`, and both players’ states are set to ‘busy’ and the second player’s `PlayerWaiting` is set to `true` to indicate that it is the other player’s turn.

The server sends a “START” message to indicate that a match has started as well as a “NAME” message with the name of the opposing player to each of the two players.

`playerExited` is set to `False` to indicate that no player has previously left that match, and the game’s `isActive` is set `True`.

While a game is ongoing

A while loop runs while `isActive` is `True`. At the start of an iteration of the loop, if `killThread` is `true` (due to a player leaving), the loop stops and exits.

If `playerExited = True`, indicating that a player left the game, a “LEFT” message is sent to the remaining player and sets the player to be first with ‘X’ pieces and waits for a second client to connect.

The server sends a “DISPLAY” message with a visualization of the board to both of the clients and calls `game.checkLoser()` to see if the losing or tying condition has been met. If it has, the server will send the appropriate protocol message to each of the clients and then restarts the game.

Otherwise, the server checks which player’s turn it is and sends a “GO” message to that player.

Handling a ‘login’ command during a game

If the server receives a “LOGIN” request during a game, it ignores the request and replies with an “ERROR” message.

Handling a ‘place’ command during a game

If the server receives a “PLACE” message during a game and it is not the sending client’s turn, it is ignored and the server replies with an “ERROR” message.

If the server receives a “PLACE” message from the client whose turn it is and the position is invalid, the server replies with an “ERROR” message.

If the position is valid, the server updates the game board, responds with an “OK” message, and sets the turn to the other player’s.

Handling an ‘exit’ command during a game

If the server receives an “EXIT” message during a game, the server sends an “OK” message back to the client that sent it, removes the player from the `playerList` and the game object, sets `killThread` to `True` to indicate that a player has left, and decrements the number of connections.

Handling all other messages or commands during a game

If the server receives a message that is not any of the previously mentioned commands, it sends an “ERROR” message back to the client that sent it.

Handling if the other player has left the game

If the other player has left the game, the remaining player is put back into a loop waiting for another player to connect.

CONTROL FLOW

Initializing server

The server has two static variables for the hostname and port.

The server then passes these two variables as arguments to construct a ThreadedTCPServer object that creates a ThreadedTCPHandler each time a client connects to the server.

The server then uses “`server_thread = threading.Thread(target=server.serve_forever)`” to enable multithreading and “`server_thread.start()`” to begin multithreading.

Receiving a connection

Upon receiving a connection from a client to the localhost port 1337, ThreadedTCPServer creates a new thread associated with the client and creates an instance of the ThreadedTCPHandler object. The thread runs the “`handle(self)`” method from the ThreadedTCPHandler object, which handles all the messages received from the client.

Terminating connections

If a player exits, the client’s associated thread will update the game status in the “`handler`” function and then terminates the thread.

SERVERP2.PY

*** This is the server for Part 2 of the final project. It shares many of the same assets as serverp1.py, but has the following differences:

GLOBAL VARIABLES

connections	***REMOVED***
localPlayerList	temporary list of players used during the execution of 'play'
gameList	list of active games
gameIndexList	list of every game that existed
totalGames	a count of how many active games have been started on the server

CLASSES

ThreadedTCPHandler

FUNCTIONS

findGameByID(self, gameID)

Player

VARIABLES

connSocket	the connection socket the client used to connect to the server
------------	--

METHODS

getConnSocket	returns connSocket
setConnSocket	sets connSocket

Game

VARIABLES

gameID	ID for the match
--------	------------------

METHODS

getGameID	returns gameID
setGameID	sets gameID

CODE

handle(self)

Variables

localGameID	ID for the game running on the thread
lobbyLoop	flag for controlling whether a client is in the lobby
exitLobbyLoop	flag to allow a client to exit the lobby

Accepting incoming connections

Unlike in serverp1.py, the server does not prevent more than two clients from connecting and allows an indefinite amount of connections.

Handling commands or messages before a game has begun

If a client has connected but has not logged in and sends a “WHO” request to the server, the server responds with an “OK” message that contains a list of all connected players’ names.

If a client has connected but has not logged in and sends a “GAME” request to the server, the server responds with an “OK” message that contains a list of all active game IDs.

If a client has connected but has not logged in and sends a “PLAY” request to the server, the server responds with an “ERROR” message.

Unlike serverp2.py, The server no longer automatically puts two players into a match.

Instead, each thread for each client enters a loop “`while lobbyLoop == True:`” which simulates a lobby and during which handles messages from the clients.

Handling a ‘login’ command in the lobby

If the server receives a “LOGIN” request from a client in the lobby loop, it ignores the request and replies with an “ERROR” message.

Handling a ‘place’ command in the lobby

If the server receives a “PLACE” request from a client in the lobby loop, it ignores the request and replies with an “ERROR” message.

Handling an ‘exit’ command in the lobby

If the server receives an “EXIT” message from a client in the lobby loop, it replies with an “OK” message and terminates the thread containing the handler.

Handling a ‘who’ command in the lobby

If the server receives a “WHO” request from a client in the lobby loop, it replies with an “OK” message with a string containing all of the logged in players’ names separated by spaces.

Handling a ‘games’ command in the lobby

If the server receives a “GAMES” request from a client in the lobby loop, it replies with an “OK” message with a string containing all the active games’ IDs and their players’ names.

Handling a ‘play’ command in the lobby

If the server receives a “PLAY” request with the target player name from a client in the lobby loop, it checks if a player exists with the given name. If it does not exist, it responds with an “ERROR” message. Otherwise, the client that sent the request has its lobbyLoop set to False, letting it exit the lobby loop, sends an “OK” message to the client, and sends a “MATCHED” message to the target player. The server then sets the requesting player’s piece to ‘X’ and is its turn.

Handling an ‘OK’ message in the lobby

If the server receives an “OK” message from a client in the lobby loop, then a player has received a “MATCHED” message after another player sent a “PLAY” request. The player’s piece is set to ‘X’ and it is not its turn, and sets its lobbyLoop to False to let it exit the lobby loop.

Starting a game

After two players have been matched, a game object and its assets are instantiated, and the global totalGames variable is incremented. The game’s ID is then set to totalGames’s value. Since totalGames is never decremented, no two games will have the same ID.

The server then sends the game's ID as well as the opposing player's name to both players.

Handling if another player left the game

If an opposing player left the match, then the remaining client will be put back into the lobby loop.

Handling a 'who' command during a game

If the server receives a "WHO" request from a client in a game, it replies with an "OK" message with a string containing all of the logged in players' names separated by spaces.

Handling a 'games' command during a game

If the server receives a "GAMES" request from a client in a game, it replies with an "OK" message with a string containing all the active games' IDs and their players' names.

Handling a 'play' command during a game

If the server receives a "PLAY" request from a client in a game, it ignores it and replies with an "ERROR" message.

findGameByID(self, gameId)

Retrieves a game object given the game ID

Searches through the gameList for a specified game ID. If there is a match, it returns the game object with the given game ID.

REVERSE TIC-TAC-TOE CLIENT

CLIENTP1.PY

*** This is the client for Part 1 of the final project

LANGUAGE

Python 3.5

MODULES

sys	used to obtain command line arguments at compilation
socket	used to connect the client to a server socket to exchange messages

GLOBAL VARIABLES

The client contains a global variable for each of the protocol methods and status codes, as well as two variables for the port and hostname:

PORT	the port number that the server is listening at. Defaults to '1337'
loggedIn	a flag to indicate whether the client is logged into a server. Defaults to False

FUNCTIONS

main()	this is the main function of the client, and is the only function executed by the client
---------------	--

CODE

main()

Checking the number of command line arguments

“`len(sys.argv)`” returns the number of arguments passed into the command line at run time. The python script must be run with 3 arguments: the python file name, the hostname of the computer running the server program, and the port the server is listening at. Any other number of arguments will print an error and terminate the program.

Checking the validity of the command line arguments

`sys.argv[1]` is the argument following the python file name. This is expected to be the hostname of the server.

`sys.argv[2]` is the second argument following the python file name. This is expected to be the port number that the server is listening at. In this implementation, the server's port is always '1337'.

`try: portNumber = int(sys.argv[2])` converts the port number argument from a string to an integer. If the argument is not a number, then an error is thrown and caught by `except ValueError:`, which prints an error message and terminates the program. If the argument is not '1337', then an error message is printed and the program is terminated.

`clientSocket = socket(AF_INET, SOCK_STREAM)` initializes the client socket that will be used to establish a connection with the server.

`clientSocket.connect((HOST, PORT))` connects the client socket to the server. If the connection is refused, the server hostname entered is invalid and an error message is printed and the program terminates.

Connecting to the server

`clientSocket.recv(1024).decode().split()` decodes a message received from the connected server and splits it into a list separated by spaces. If the first entry in the list is the protocol number for "OK", the server has accepted the connection. Otherwise, the connection is rejected and the client program terminates.

Main loop for handling input

An indefinitely running loop that handles all input from the player as well as all messages received from the server.

`arguments = input("> ").split()` prints "> " to the command line and waits for the user to enter a command into the command line, which is split into a list separated by spaces named 'arguments'.

Empty input

If the input is empty, then the loop begins again. Otherwise, the loop checks the input to determine if it is a valid command and acts accordingly.

Handling the ‘help’ command

Checks if the input is “help” and prints the ‘help’ message if true.

Handling the ‘login’ command

Checks if the input is “login” followed by an argument (the username).

If true, the ‘login’ command is executed.

“loginMessage = LOGIN + " " + arguments[1]” creates a string named ‘loginMessage’ with the LOGIN protocol and the player’s username.

“clientSocket.send(loginMessage.encode())” encodes and sends the login protocol to the server.

“response = clientSocket.recv(1024).decode()” waits for and receives a response from the server, and then decodes the message.

“tokenized = response.split()” splits the decoded response message into a list separated by spaces. The first entry in the list is then checked to see if an “OK” protocol message is returned, which will then continue the loop, or if an “ERROR” protocol message is returned, which will print an error if the username is invalid or if the player is already logged in, and the loop will begin again.

If the message is an “OK” protocol message, then the global variable loggedIn is set True and the client then waits for another response from the server.

If the next message received is a “WAIT” protocol message, then there are not enough players to begin a match and the client waits until it receives a “START” protocol message to indicate that a match has begun.

Once the “START” protocol message has been received, the client waits for a “NAME” protocol message containing the name of the opponent, and prints the name of the opponent once it receives the message.

Then the client then waits for a “DISPLAY” protocol message containing the contents of the board, after which it will notify the player that the game has begun and will print out the board and waits for the next response.

If the next response is a “GO” protocol message, the client will notify the user that it is his turn and the loop will restart to handle the user’s next input.

Otherwise, if the response is a “WAIT” protocol message, the client will notify the user that it is the other player’s turn.

The client waits for a “DISPLAY” protocol message and updates and prints the board upon receiving it and waits for the next response from the server.

If the next response is a “GO” protocol message, the loop will restart to handle the user’s next input.

If the next response is a “LEFT” protocol message, the opponent left the match and the client waits for another match to begin.

Handling the ‘place’ command

Checks if the input is “place” followed by an argument (the position on the board to place a game piece). If true, the ‘place’ command is executed. “`tileNumber = int(arguments[1])`” converts the argument to an integer. If the argument is not a number, or is not a number in the range of 1 to 9, then an error is printed and the loop will restart to handle the user’s next input.

If the command is valid, the client uses “`placeMessage = PLACE + " " + arguments[1]`” to create a “PLACE” protocol message and “`clientSocket.send(placeMessage.encode())`” sends the message to the server.

If the return message is an “ERROR” protocol message, then the loop restarts to handle the user’s next input.

Otherwise, if the server returns an “OK” protocol message, the next response message received from the server contains the updated game board.

The client then waits for the next message from the server.

If the next message is a “WAIT” message, then the client informs the player that the move was made successfully and that it is the opponent’s turn. If the next message afterwards is a “LEFT” message, then the game has ended and the client waits for another game to start.

If the message is a “WON”, “LOST”, or “TIED” protocol message, then the game has ended and the client displays whether the player had won, lost, or tied, respectively, and waits for another game to start.

If the message is a “GO” message, the client will notify that it is the player’s turn and the loop will restart to handle the user’s next input.

Handling the ‘game’ command

Checks if the input is “games” with no following arguments. If true, the ‘game’ command is executed.

`gamesMessage = GAMES` creates a “GAMES” protocol message.

`clientSocket.send(gamesMessage.encode())` sends the protocol message to the server.

The client then waits for a response message from the server.

If the next message is an “OK” protocol message, it contains the data of the other games as a list of strings containing the game ID and the two player IDs separated by spaces. If the message only contains the protocol, then there are no active games. Otherwise, the client splits each of the strings in the list, separated by spaces, and prints them.

If the message is an “ERROR” protocol message, then the client prints an error and restarts the loop.

Handling the ‘who’ command

Checks if the input is “who” with no following arguments. If true, the ‘who’ command is executed.

The client sends a “WHO” protocol message to the server and waits for a response.

If the response is an “OK” message, it contains a list of the IDs of all players ready to play. The client then prints the IDs and restarts the loop.

If the response is not an “OK” message, then the client prints an error message and restarts the loop.

Handling the ‘play’ command

Checks if the input is “play” followed by an argument (the ID of the target player). If true, the ‘play’ command is executed.

The client sends a “PLAY” protocol message to the server and waits for a response.

If the response is an “ERROR” protocol message, then the client prints an error message and restarts the loop.

If the response is an “OK” message, then the server has created a game with player and the target player, and the client restarts the loop to handle the user’s next input.

Handling all other inputs

For all other inputs, the client prints an error message to indicate that the input is invalid, and the loop restarts.

CONTROL FLOW

Running main()

Calls the main() function on execution of the python file

CLIENTP2.PY

***This is the client for Part 2 of the final project. Clientp2.py shares the same assets as clientp1.py, but has the following additions/changes:

MODULES

selectors	used to listen and send out messages simultaneously
time	used for the sleep function to put threads to sleep

GLOBAL VARIABLES

inGame	flag to indicate if the client is in a game
clientSocket	the socket that the client uses to connect to the server. Moved from local to global.
justMatched	flag to indicate if a player was just matched into a game
isChallenger	flag to indicate if the player was the one who used "PLAY" to start a match

FUNCTIONS

parseInput()	function used to handle user input
parseMessage()	function used to handle responses from the server

CODE

parseInput()

Handles the commands, 'help', 'login', 'place', 'game', 'who', and 'play' commands the same way as clientp1.py's main function did, and terminates the function upon handling the command. However, it handles game overs differently.

Receiving "WON", "LOST", "TIED", or "LEFT" messages from server

Returns a message stating that the player has been sent back to the lobby and terminates the function.

parseMessage()

Handling others attempting to play against the player

If the player is not a challenger and receives a “MATCHED” message from the server, the client sets `inGame` to `True` and `justMatched` to `True`, and sends an “OK” message to the server. If the player receives an “OKAY” message instead while `justMatched` is `True`, then it is the return message from the previously sent “OK” message, and `justMatched` is set to `False`.

A game has started

If the player is the challenger, he/she has the first turn, and is notified. Otherwise, he/she is notified to wait for the other player to end their turn. The player is notified of the other player’s player name and the board is displayed.

main()

Setting handlers

Instead of directly handling commands in the main function, the main function sets up selectors to notify `clientSocket` when reading/writing is to be done.

“`clientSelectors.register(clientSocket, selectors.EVENT_READ, parseMessage)`” sets it up so that any messages passed to `clientSocket` is handled by `parseMessage()`.

“`clientSelectors.register(sys.stdin, selectors.EVENT_READ, parseInput)`” sets it up so that any messages passed to `stdin` (the command line) is handled by `parseInput()`.

Handling commands and responses

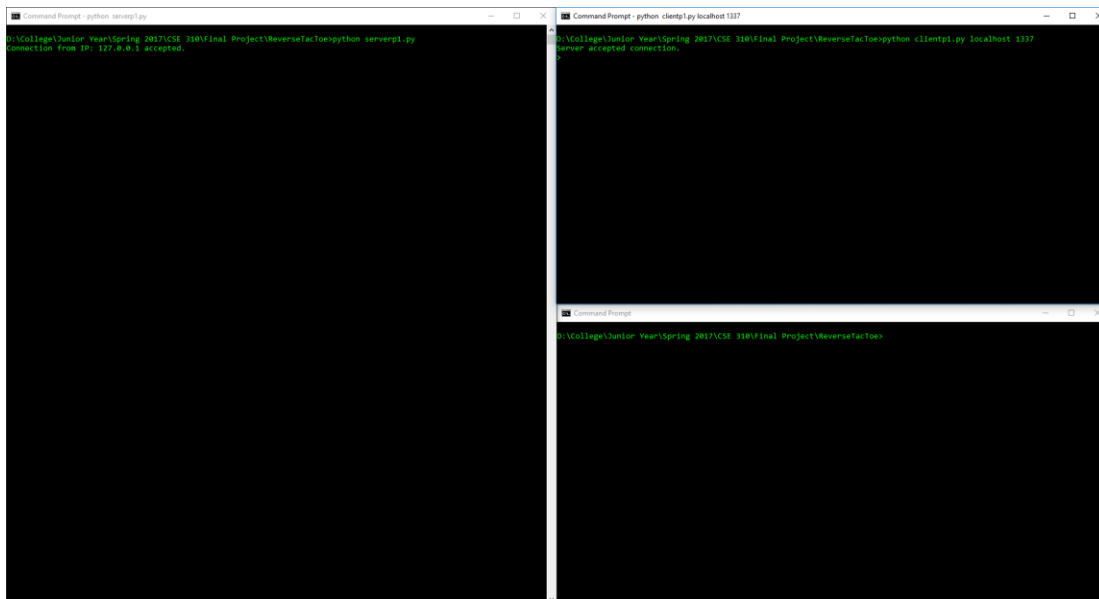
The main method has loop that runs indefinitely that triggers the selectors to handle commands and responses from the server when they are passed to the client.

=====TESTING DOCUMENTATION=====

All images document the expected output.

CLIENT-SERVER CASES (PART ONE)

NOTE: For all subsequent images, the server is on the left and the two clients are on the right.

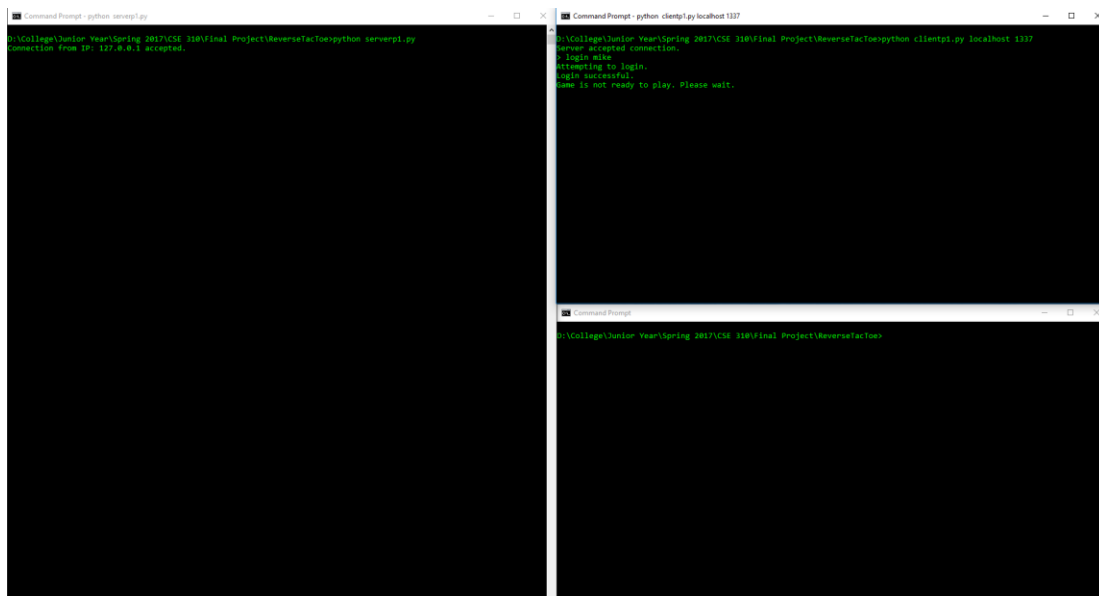


```
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactor>python server1.py
Connection from IP: 127.0.0.1 accepted.

D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactor>python client1.py localhost 1337
Server accepted connection.

D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactor>
```

Figure 1: Server started and one client successfully connected



```
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactor>python server1.py
Connection from IP: 127.0.0.1 accepted.

D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactor>python client1.py localhost 1337
Server accepted connection.
Login name
Attempting to login.
Login successful
Game is not ready to play. Please wait.

D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactor>
```

Figure 2: Connected client successfully logs in with userId: mike

The image shows two side-by-side command prompt windows. The left window, titled 'Command Prompt - python server.py', displays the following text: 'D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTaco\python server.py', 'Connection from IP: 127.0.0.1 accepted.', and 'Connection from IP: 127.0.0.1 accepted.'. The right window, titled 'Command Prompt - python client.py localhost 1337', displays: 'D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTaco\python client.py localhost 1337', 'Server accepted connection.', '> login mike', 'Attempting to login.', 'Login successful.', and 'Game is not ready to play. Please wait.'.

Figure 3: Second client successfully connected

The image shows three command prompt windows. The left window is identical to the one in Figure 3. The top-right window, titled 'Command Prompt - python client.py localhost 1337', displays: 'D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTaco\python client.py localhost 1337', 'Server accepted connection.', '> login mike', 'Attempting to login.', 'Login successful.', 'Game is not ready to play. Please wait.', 'Game is starting.', 'Your opponent's login ID is: jake', a three-line ASCII art character, and 'It is your turn. Make your move.'. The bottom-right window, also titled 'Command Prompt - python client.py localhost 1337', displays: 'D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTaco\python client.py localhost 1337', 'Server accepted connection.', '> login jake', 'Attempting to login.', 'Login successful.', 'Game is starting.', 'Your opponent's login ID is: mike', a three-line ASCII art character, and 'Opponent is making his/her move. Wait until it is your turn before inputting any commands.'.

Figure 4: Second client successfully logs in – players are auto-matched and game is started

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python client.py localhost 1337
Server accepted connection.
> login mike
Attempting to login.
Login successful.
Game is not ready to play. Please wait.
Game is starting.
Your opponent's login ID is: jake
- - -
- - -
- - -
It is your turn. Make your move.
> help
login [String]: This command takes one argument, which is your name. This name will uniquely identify you to the server. An example of how to use this command is to input 'login Michael'.
place [int]: This command takes one argument, which is an integer between 1 and 9 inclusive. This number identifies the cell that you want to occupy with this move. An example of how to use this command is to input 'place 4'.
exit: This command takes no arguments. Upon issuing this command, you will exit the server. The only way to use this command is to input 'exit'.
>

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: mike
- - -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
```

Figure 5: Help command is used mid-game

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python client.py localhost 1337
Server accepted connection.
> login mike
Attempting to login.
Login successful.
Game is not ready to play. Please wait.
Game is starting.
Your opponent's login ID is: jake
- - -
- - -
- - -
It is your turn. Make your move.
> help
login [String]: This command takes one argument, which is your name. This name will uniquely identify you to the server. An example of how to use this command is to input 'login Michael'.
place [int]: This command takes one argument, which is an integer between 1 and 9 inclusive. This number identifies the cell that you want to occupy with this move. An example of how to use this command is to input 'place 4'.
exit: This command takes no arguments. Upon issuing this command, you will exit the server. The only way to use this command is to input 'exit'.
> place 2
Attempting to place piece.
Your piece was placed.
- X -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: mike
- - -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- X -
- - -
- - -
It is your turn. Make your move.
>
```

Figure 6: First client makes a move and the board is updated to reflect it

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactox\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
Game is starting.
Your opponent's login ID is: jake
- - -
- - -
- - -
It is your turn. Make your move.
> help
login [String]: This command takes one argument, which is your name. This name will uniquely identify you to the server. An example of how to use this command is to input 'login Michael'.
place [int]: This command takes one argument, which is an integer between 1 and 9 inclusive. This number identifies the cell that you want to occupy with this move. An example of how to use this command is to input 'place 4'.
exit: This command takes no arguments. Upon issuing this command, you will exit the server. The only way to use this command is to input 'exit'.
> place 2
Attempting to place piece.
Your piece was placed.
- X -
- - -
- X -
- O -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- X -
- O -
- - -
It is your turn make your move.
>

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactox\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
login successful.
Game is starting.
Your opponent's login ID is: mike
- - -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- X -
- O -
- - -
It is your turn. Make your move.
> place 5
Attempting to place piece.
Your piece was placed.
- X -
- O -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- - -
- - -
- - -
```

Figure 7: Second client makes a move and the board is updated to reflect it

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactox\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
It is your turn make your move.
> place 1
Attempting to place piece.
Your piece was placed.
- X X -
- O -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- X X -
- O -
- - -
It is your turn make your move.
> place 3
Attempting to place piece.
Your piece was placed.
- X X X -
- O -
- - -
The game is over. You lost. Better luck next time!
Next game is starting.
- - -
- - -
- - -
It is your turn. Make your move.
>

Command Prompt - python client.py localhost 1337
- X -
- O -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- X X -
- O -
- - -
It is your turn make your move.
> place 4
Attempting to place piece.
Your piece was placed.
- X X -
- O -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
- X X X -
- O -
- - -
The game is over. You won! Congratulations!
Next game is starting.
- - -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
```

Figure 8: Game is finished – first client loses. Game then restarts

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
O | O |
It is your turn make your move.
> place 5
Attempting to place piece.
Your piece was placed.
X | O |
X | X |
O | X |
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
X | O |
X | X |
O | X |
It is your turn make your move.
> place 3
Attempting to place piece.
Your piece was placed.
X | O | X
X | X | O
O | X | O
The game ended in a tie.
Next game is starting.
+ + +
+ + +
+ + +
It is your turn. Make your move.
>

Command Prompt - python client.py localhost 1337
X | O |
X | X |
O | O |
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
X | O |
X | X |
O | X |
It is your turn make your move.
> place 6
Attempting to place piece.
Your piece was placed.
X | O |
X | X |
O | X |
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
X | O | X
X | X | O
O | X | O
The game ended in a tie.
Next game is starting.
+ + +
+ + +
+ + +
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
```

Figure 9: Game is finished – clients tie. Game is then restarted

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python server.py
Connection from IP: 127.0.0.1 accepted.

Command Prompt
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe\python client.py localhost 1337
Server accepted connection.
> exit
Exit acknowledged.
Exiting now. See you next time!
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe>

Command Prompt
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseTactoe>
```

Figure 10: First client immediately exits after connecting

The figure consists of three terminal windows. The top-left window shows the server running and accepting connections from 127.0.0.1. The top-right window shows a client logging in as 'mike', starting a game with 'jake' as the opponent, and then exiting. The bottom window shows a new client logging in as 'jake', attempting to log in with 'jake' (which fails), and then logging in successfully with 'mike' as the opponent.

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login mike
Attempting to login.
Login successful.
Game is not ready to play. Please wait.
Game is starting.
Your opponent's login ID is: jake
- - -
- - -
- - -
It is your turn. Make your move.
> exit
Attempting to exit.
Exit acknowledged.
Exiting now. See you next time!
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe>

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: mike
- - -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
Your opponent left the game. Sorry about that. Please wait while we find you a new opponent.
```

Figure 11: First client exits mid-game – second client waits for new opponent

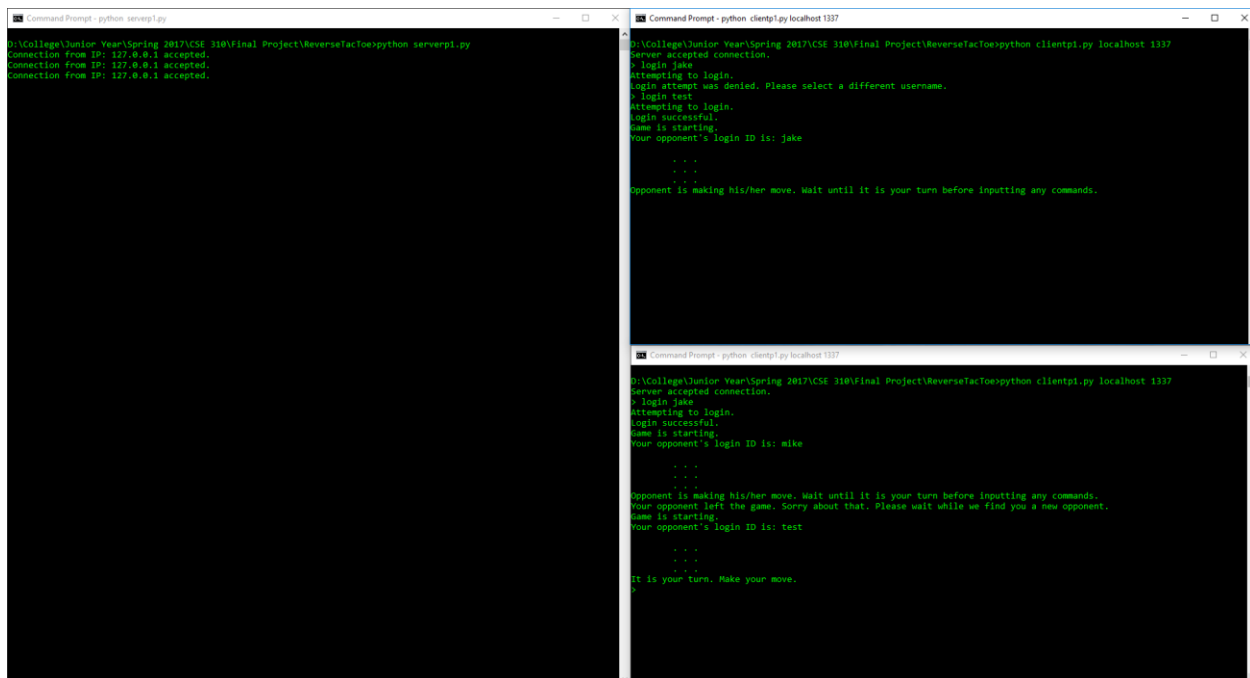
The figure consists of three terminal windows. The top-left window shows the server running and accepting connections from 127.0.0.1. The top-right window shows a client logging in as 'jake', attempting to log in with 'jake' (which fails), and then logging in successfully with 'mike' as the opponent. The bottom window shows a new client logging in as 'jake', attempting to log in with 'jake' (which fails), and then logging in successfully with 'mike' as the opponent.

```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login attempt was denied. Please select a different username.

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: mike
- - -
- - -
- - -
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
Your opponent left the game. Sorry about that. Please wait while we find you a new opponent.
```

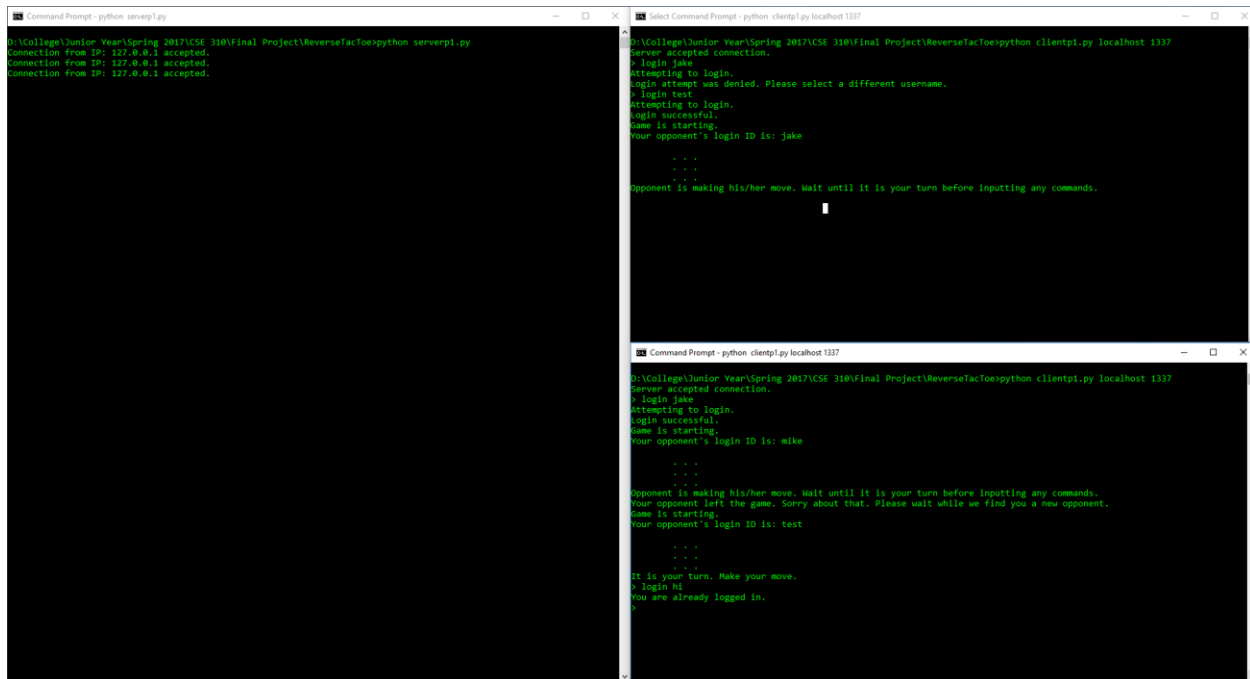
Figure 12: First client unsuccessfully tries to login with a username already in use



```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login attempt was denied. Please select a different username.
> login test
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: jake
+ + +
+ + +
+ + +
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
>
```

Figure 13: First client logs in and is auto-matched with existing player (after initial exit)



```
Command Prompt - python server.py
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python server.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.

Select Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login jake
Attempting to login.
Login attempt was denied. Please select a different username.
> login test
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: jake
+ + +
+ + +
+ + +
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
>
```

```
Command Prompt - python client.py localhost 1337
D:\College\Junior Year\Spring 2017\CSE 310\Final Project\ReverseFactoe\python client.py localhost 1337
Server accepted connection.
> login hi
Attempting to login.
Login successful.
Game is starting.
Your opponent's login ID is: mike
+ + +
+ + +
+ + +
Opponent is making his/her move. Wait until it is your turn before inputting any commands.
Your opponent left the game. Sorry about that. Please wait while we find you a new opponent.
Game is starting.
Your opponent's login ID is: test
+ + +
+ + +
+ + +
It is your turn. Make your move.
> login hi
You are already logged in.
>
```

Figure 14: Second client unsuccessfully tries to login while already logged in

CLIENT-SERVER DOCUMENTATION (PART TWO)

NOTE: For all subsequent images, the server is in the top-left and the clients are the other prompts.

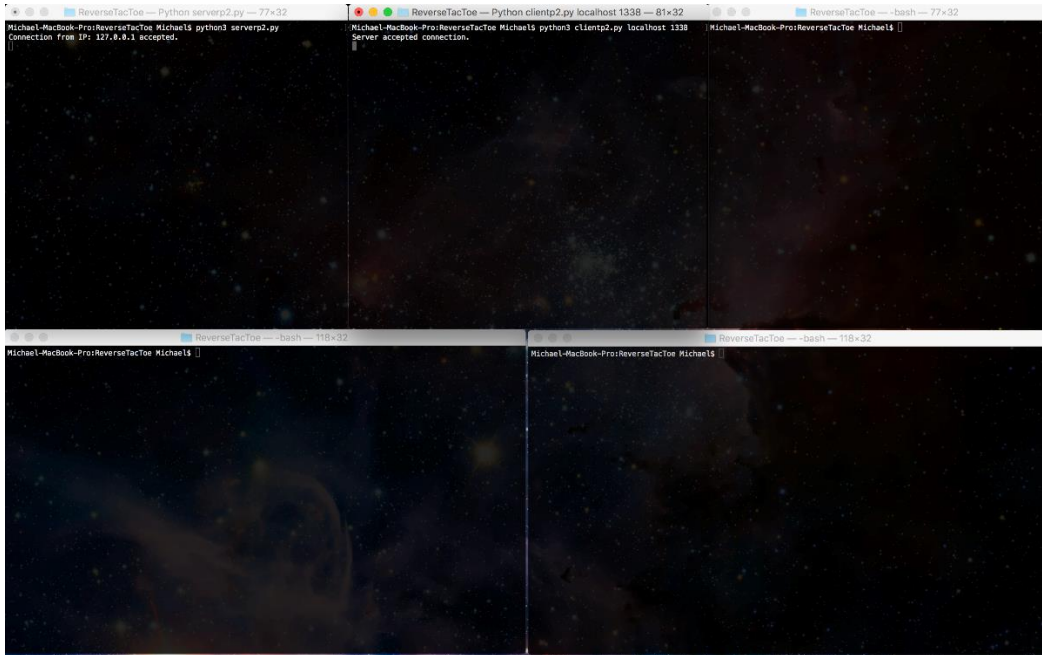


Figure 1: Server successfully accepts connection from first client.

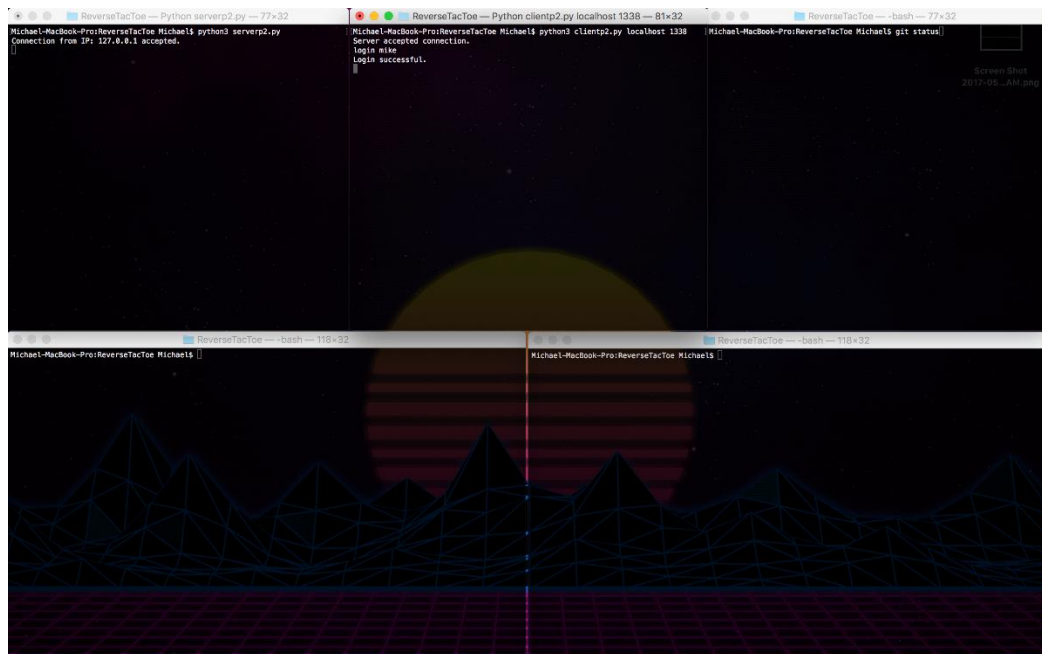


Figure 2: First client successfully logs in and is placed in lobby

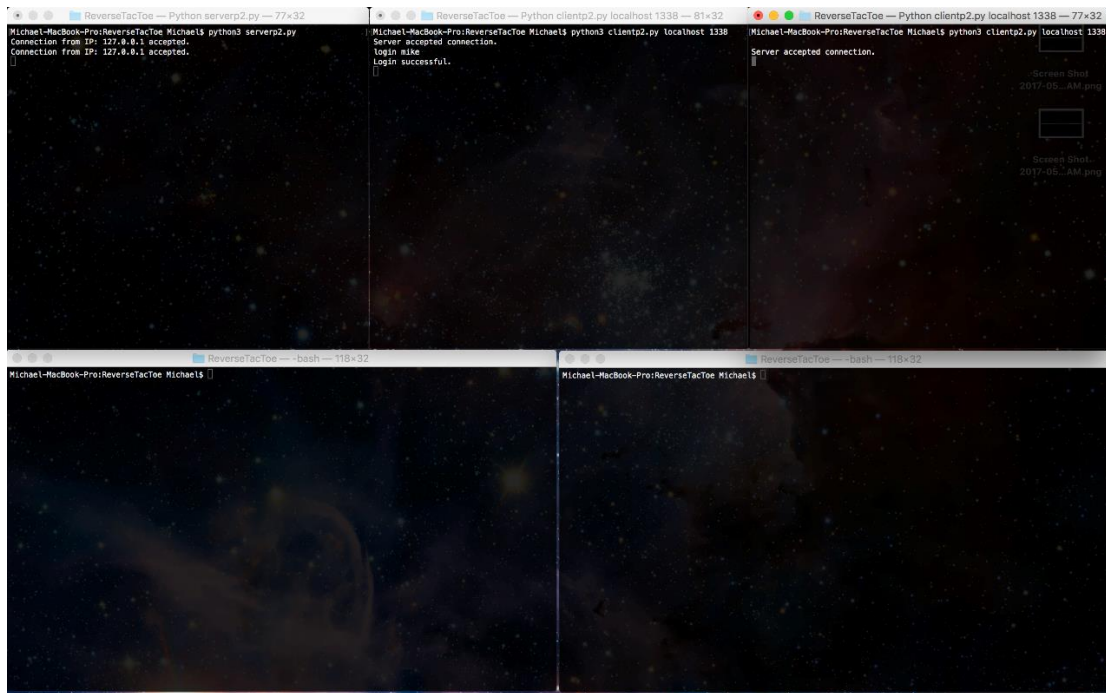


Figure 3: Server successfully accepts connection from second client

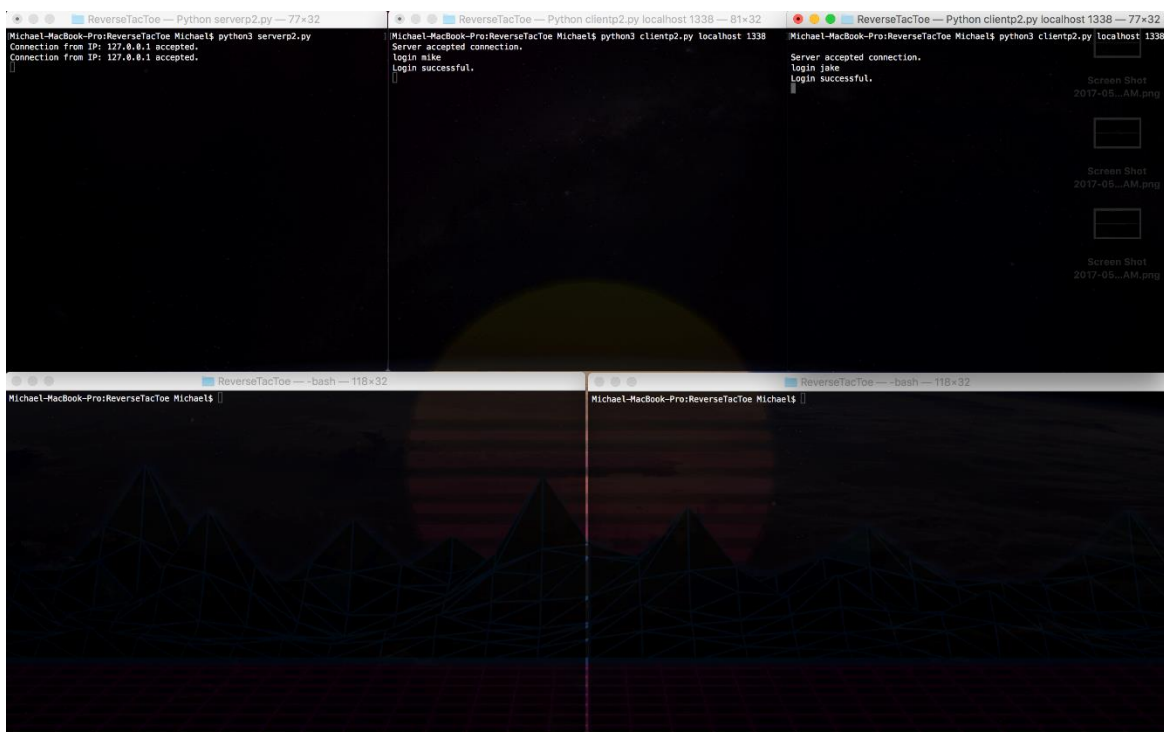


Figure 4: Second client successfully logs in and is placed in lobby

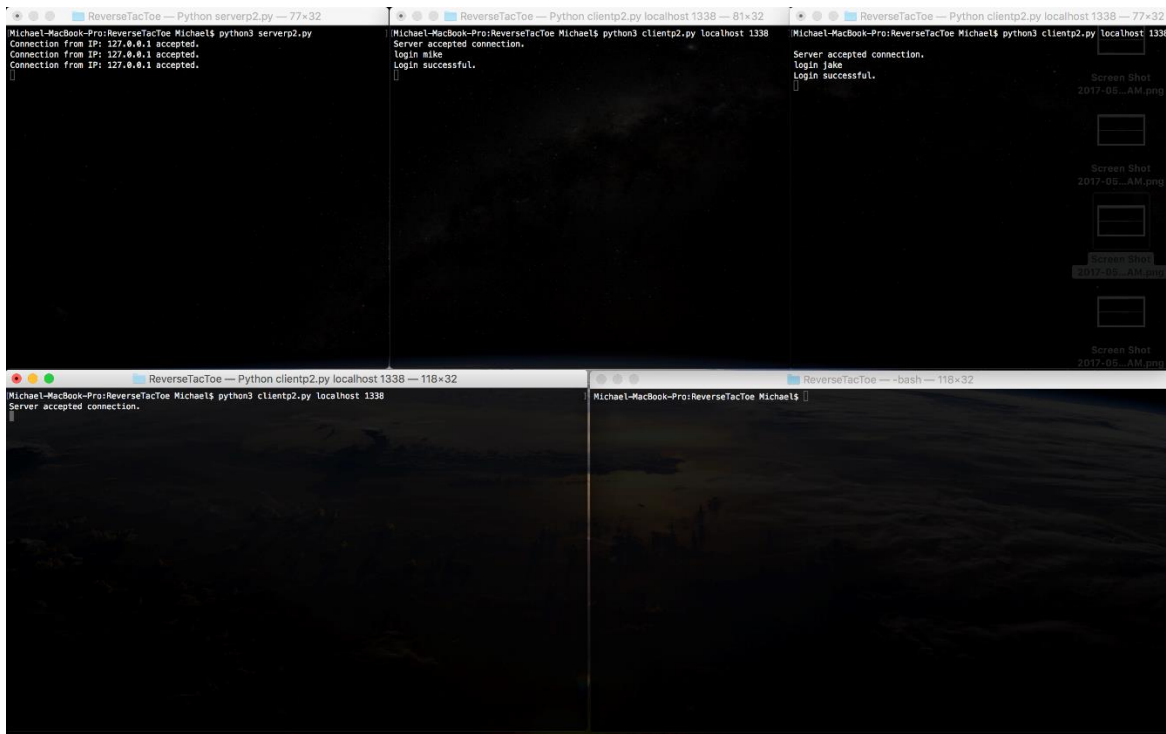


Figure 5: Server successfully accepts connection from third client

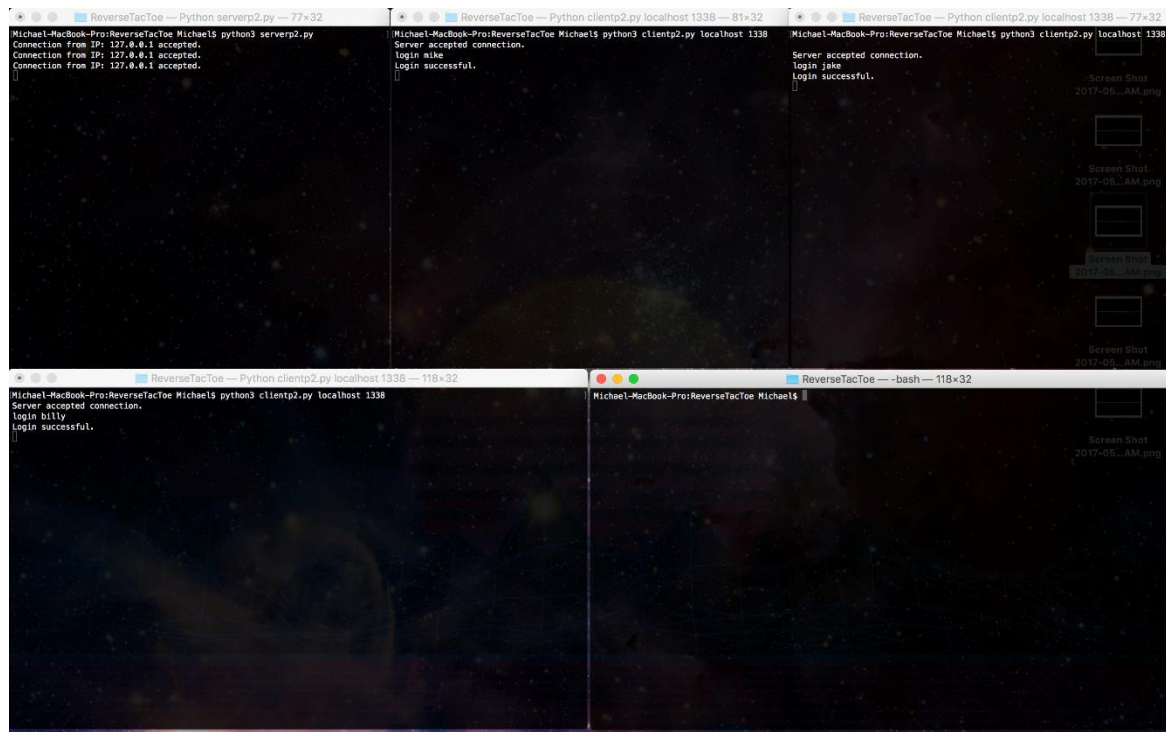


Figure 6: Third client successfully logs in

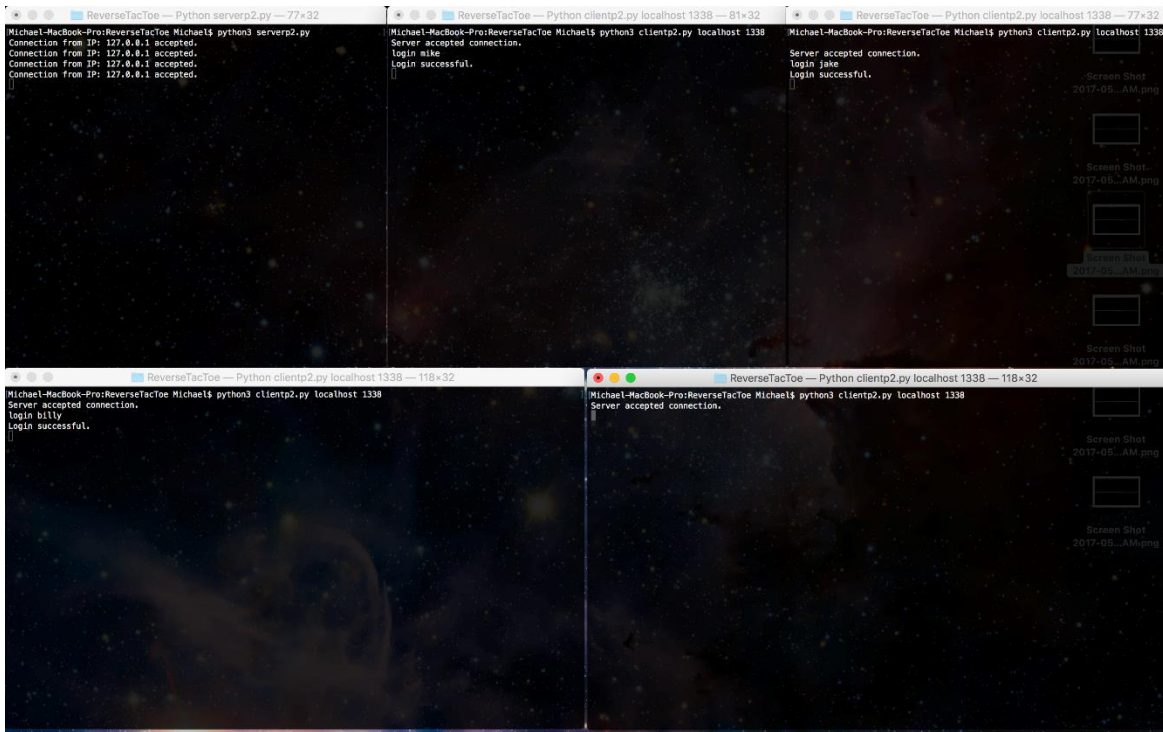


Figure 7: Server successfully accepts connection from fourth client



Figure 8: Fourth client successfully logs in

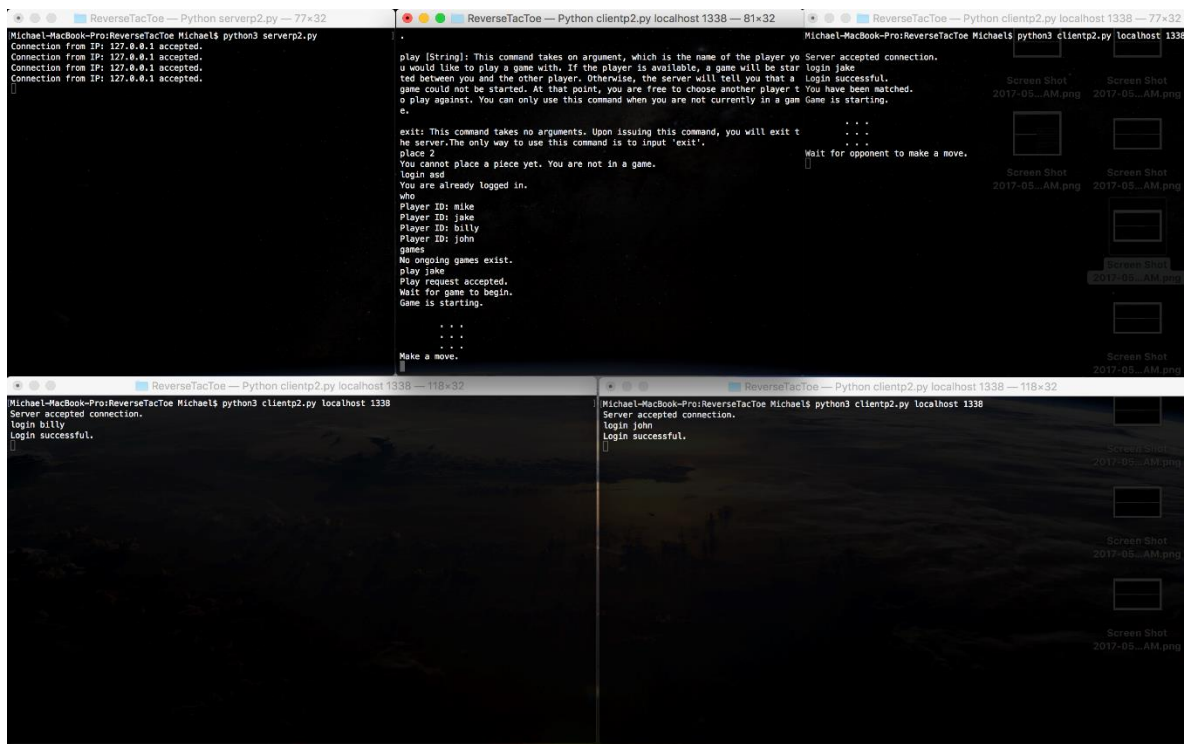


Figure 9: First client challenges second client (“play”) and a game is setup between them

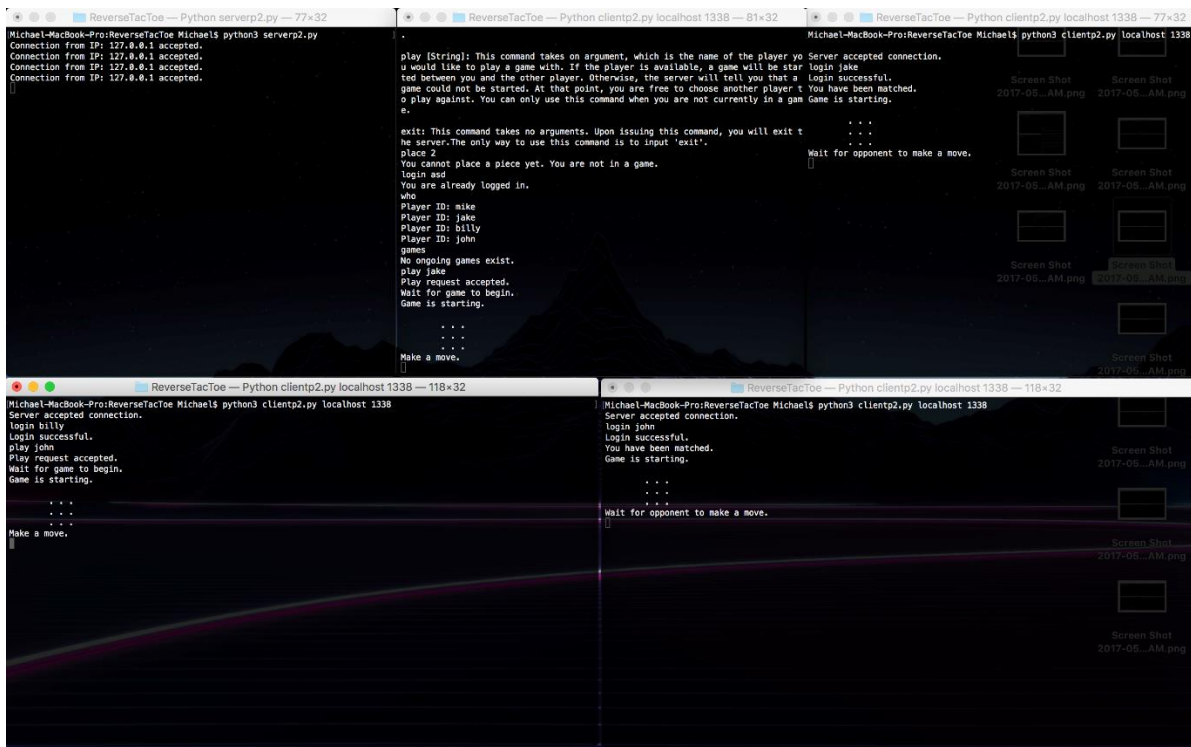


Figure 10: Third client challenges fourth client (“play”) and a game is setup between them

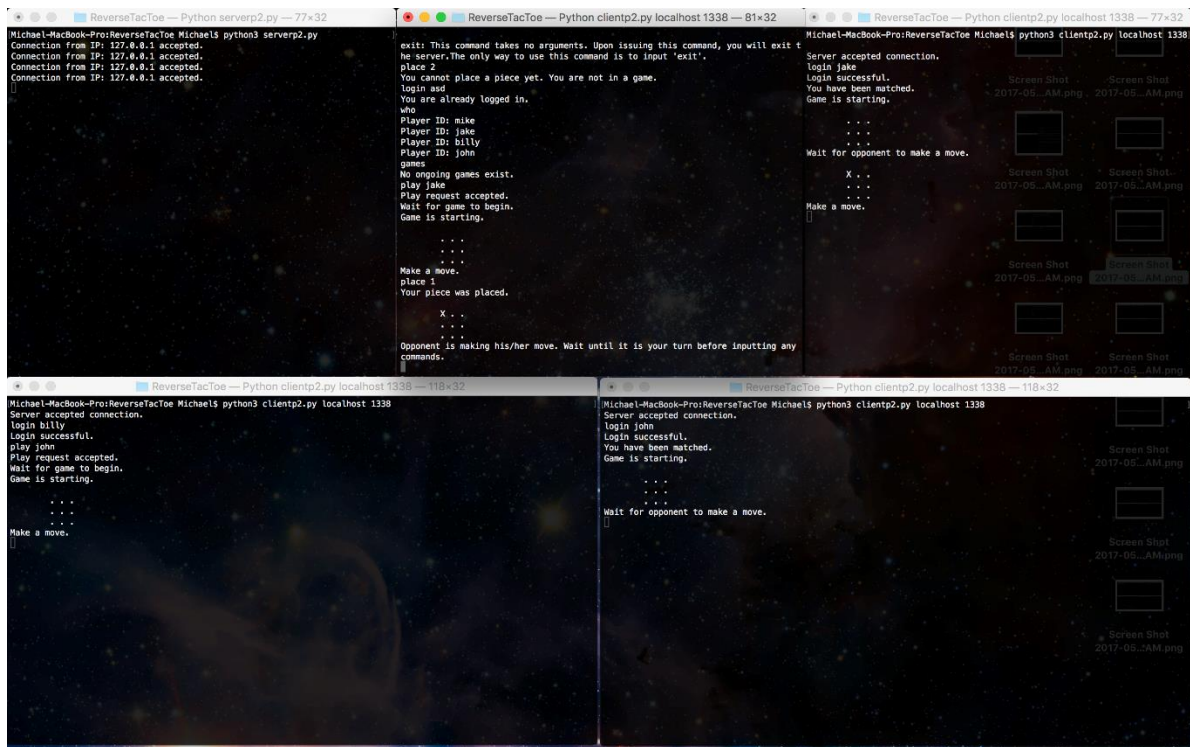


Figure 11: First client makes a move ("place") in his game – the board is then updated

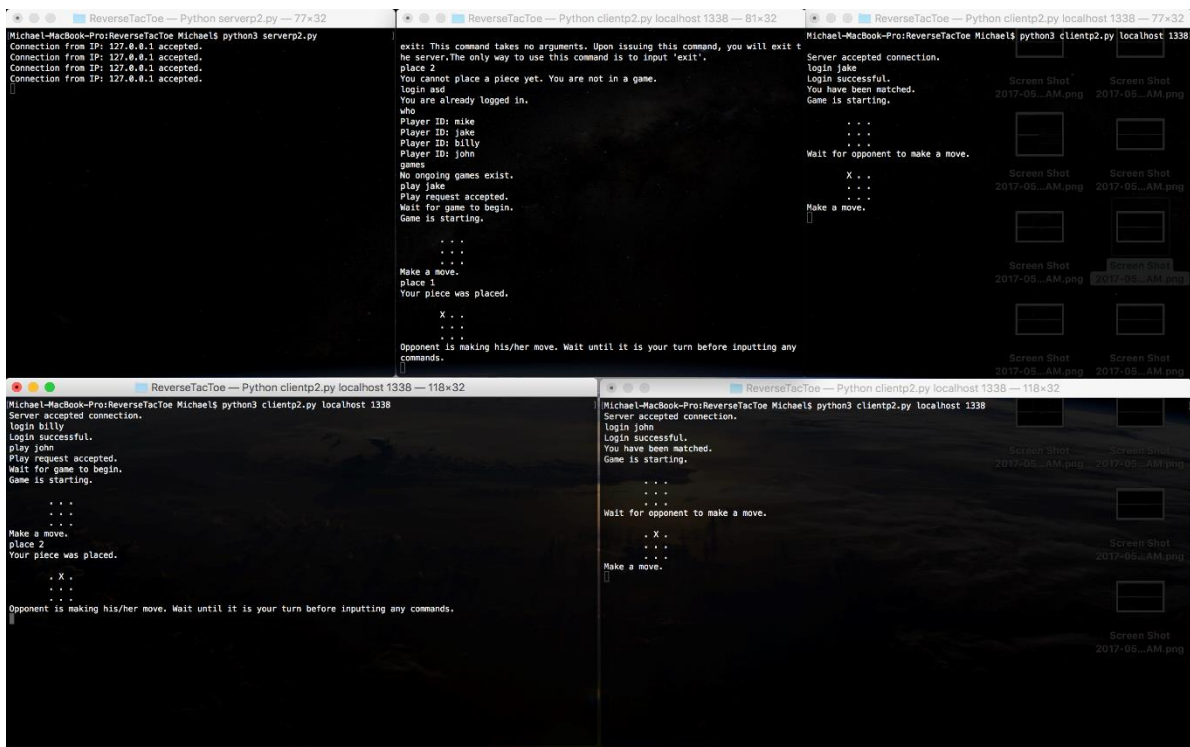


Figure 12: Third client makes a move ("place") in his game – the board is then updated

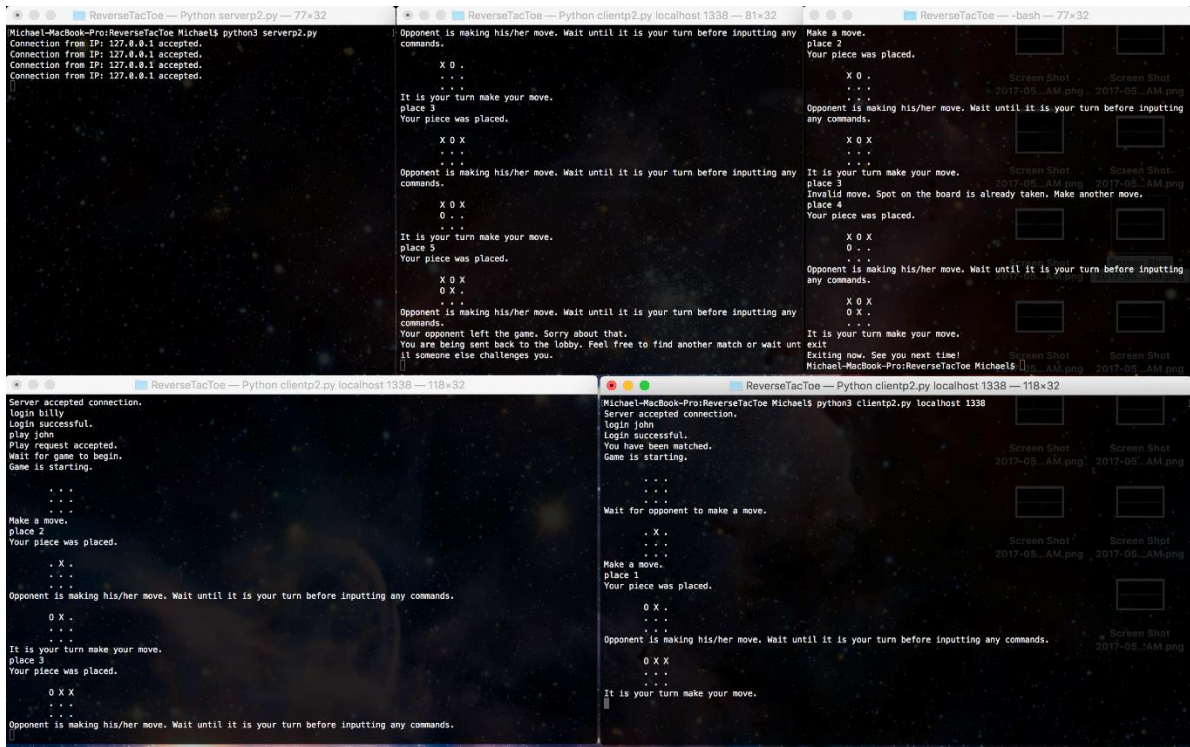


Figure 13: Second client exits his game – first client's game is ended and he is put back in a lobby

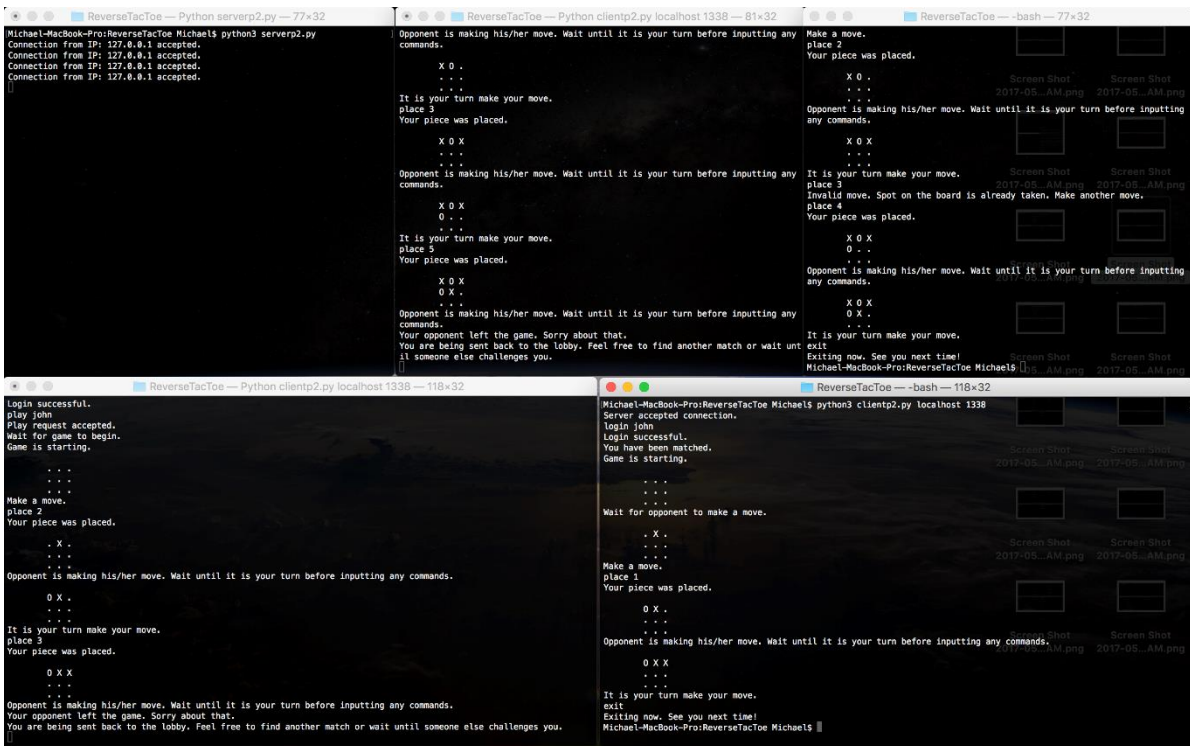


Figure 14: Fourth client exits his game – third client's game is ended and he is put back in a lobby

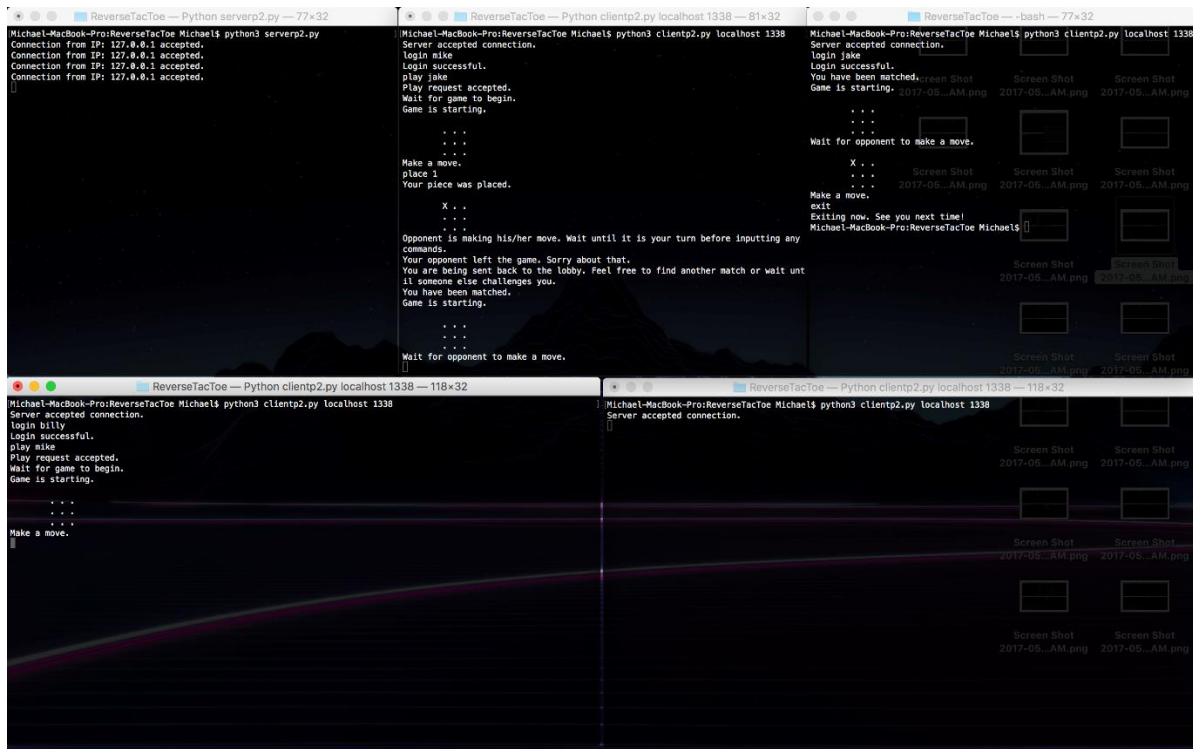


Figure 15: Demonstrates the ability to have a game be ended and then challenge someone else

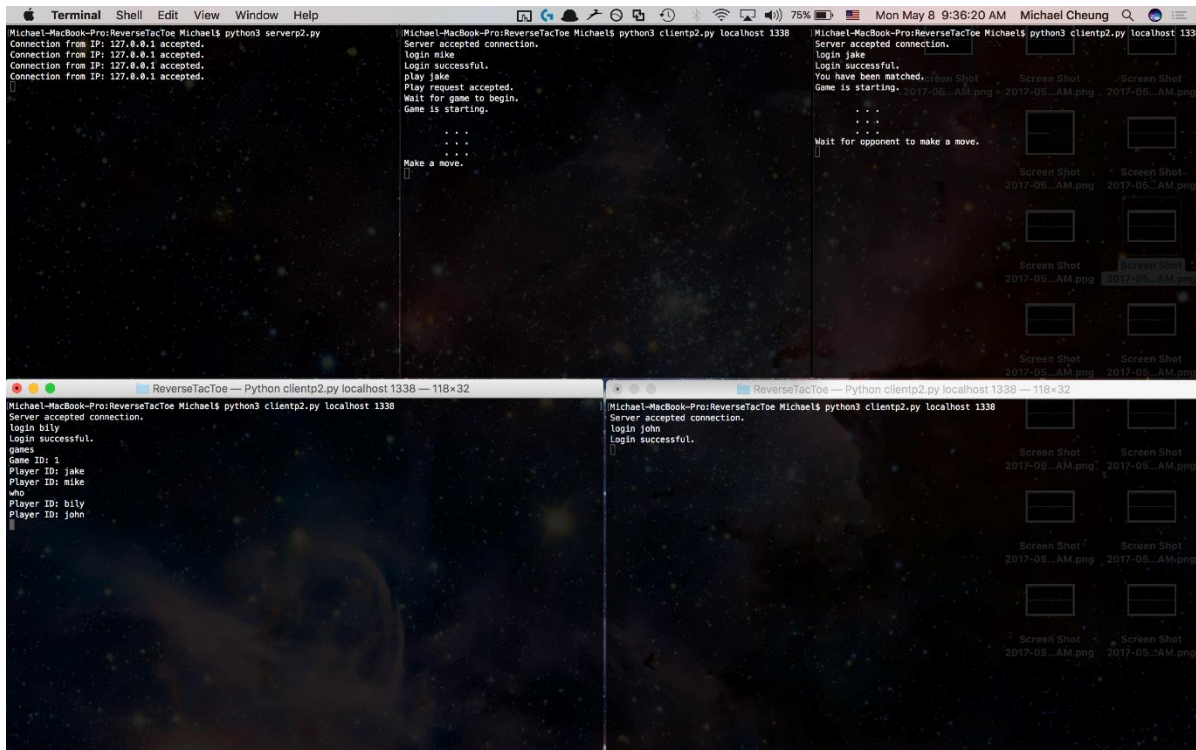


Figure 16: Demonstrates “game” and “who” commands

```
ReverseTacToe — Python serverp2.py — 77x32
Michael-MacBook-Pro:ReverseTacToe Michael$ python3 serverp2.py
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.
Connection from IP: 127.0.0.1 accepted.
[]

ReverseTacToe — Python clientp2.py localhost 1338 — 81x66
Michael-MacBook-Pro:ReverseTacToe Michael$ clear

Michael-MacBook-Pro:ReverseTacToe Michael$ python3 clientp2.py localhost 1338
Server accepted connection.
login mike
Login successful.
help
login [String]: This command takes one argument, which is your name. This name will uniquely identify you to the server. An example of how to use this command is to input 'login Michael'.

place [int]: This command takes one argument, which is an integer between 1 and 9 inclusive. This number identifies the cell that you want to occupy with this move. An example of how to use this command is to input 'place 4'.

games: This command takes no arguments. It will trigger a query that is sent to the server, which will return a list of current ongoing games. The game ID and the players are listed per game. You can only use this command when you are not currently in a game.

who: This command takes no arguments. It will trigger a query that is sent to the server, which will return a list of players that are currently logged-in and available to play. You can only use this command when you are not currently in a game.

play [String]: This command takes one argument, which is the name of the player you would like to play a game with. If the player is available, a game will be started between you and the other player. Otherwise, the server will tell you that a game could not be started. At that point, you are free to choose another player to play against. You can only use this command when you are not currently in a game.

exit: This command takes no arguments. Upon issuing this command, you will exit the server. The only way to use this command is to input 'exit'.
place 2
You cannot place a piece yet. You are not in a game.
login asd
You are already logged in.
who
Player ID: mike
Player ID: jake
Player ID: billy
Player ID: john
games
No ongoing games exist.
[]
```

Figure 17: Demonstrates commands running when not logged in