INSTRUCTIONS TO RUN

These programs must be run with python3. They can be opened and run from within the PyCharm IDE, or from the command line by typing: python path/chat_game_server.py or python path/chat_game_client.py, respectively.

The server must be started first so that the client is able to connect. Once both programs are started, connection is initiated from the client side by sending a non-empty message (the console provides instructions).

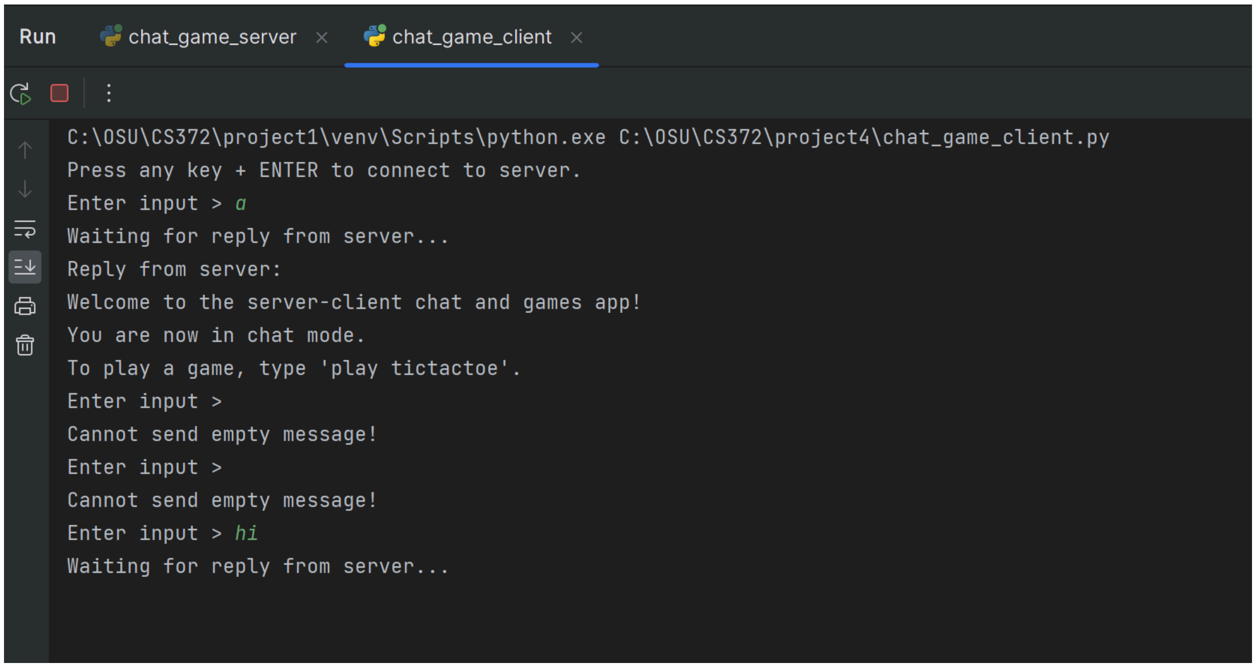
This program uses the socket and math libraries.

SOURCES CITED

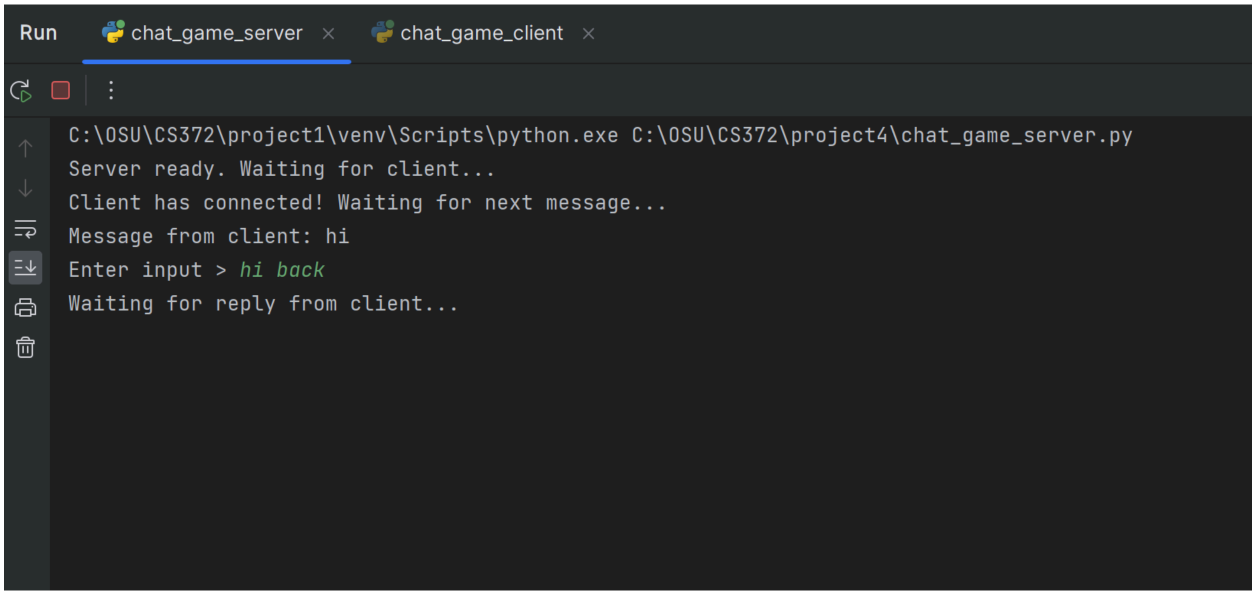
n/a

SCREENSHOTS

1. Chatting functionality, server- and client-side:

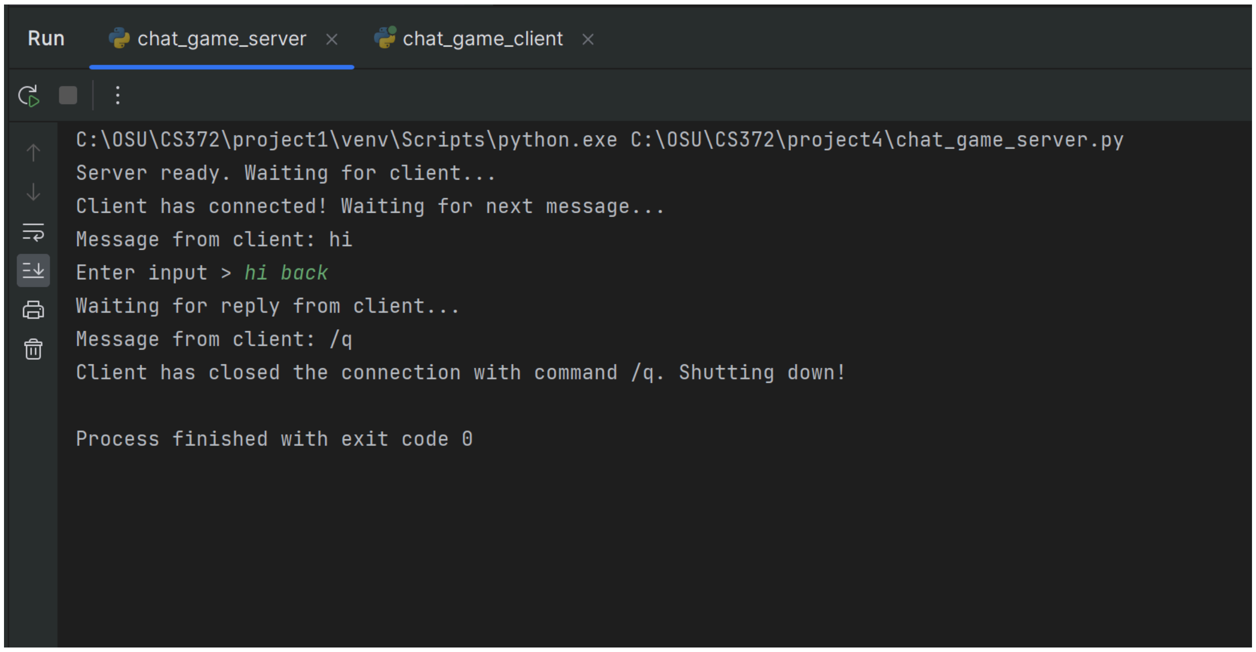


```
C:\OSU\CS372\project1\venv\Scripts\python.exe C:\OSU\CS372\project4\chat_game_client.py
Press any key + ENTER to connect to server.
Enter input > a
Waiting for reply from server...
Reply from server:
Welcome to the server-client chat and games app!
You are now in chat mode.
To play a game, type 'play tictactoe'.
Enter input >
Cannot send empty message!
Enter input >
Cannot send empty message!
Enter input > hi
Waiting for reply from server...
```

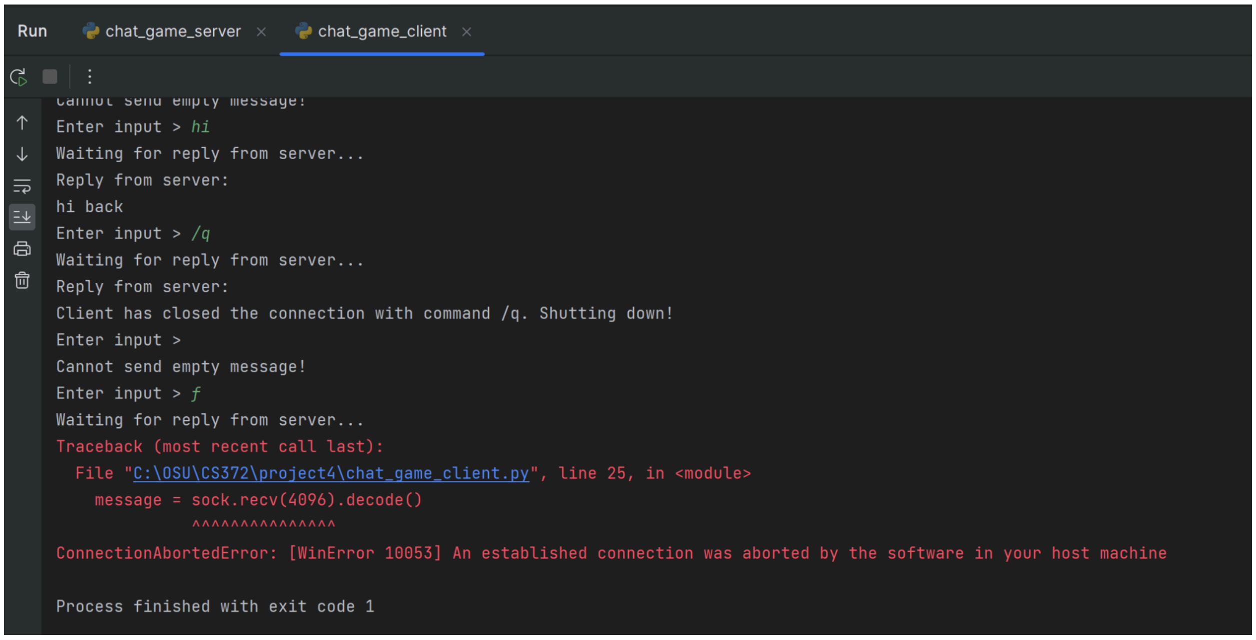


```
C:\OSU\CS372\project1\venv\Scripts\python.exe C:\OSU\CS372\project4\chat_game_server.py
Server ready. Waiting for client...
Client has connected! Waiting for next message...
Message from client: hi
Enter input > hi back
Waiting for reply from client...
```

2. Quitting the program from either side:

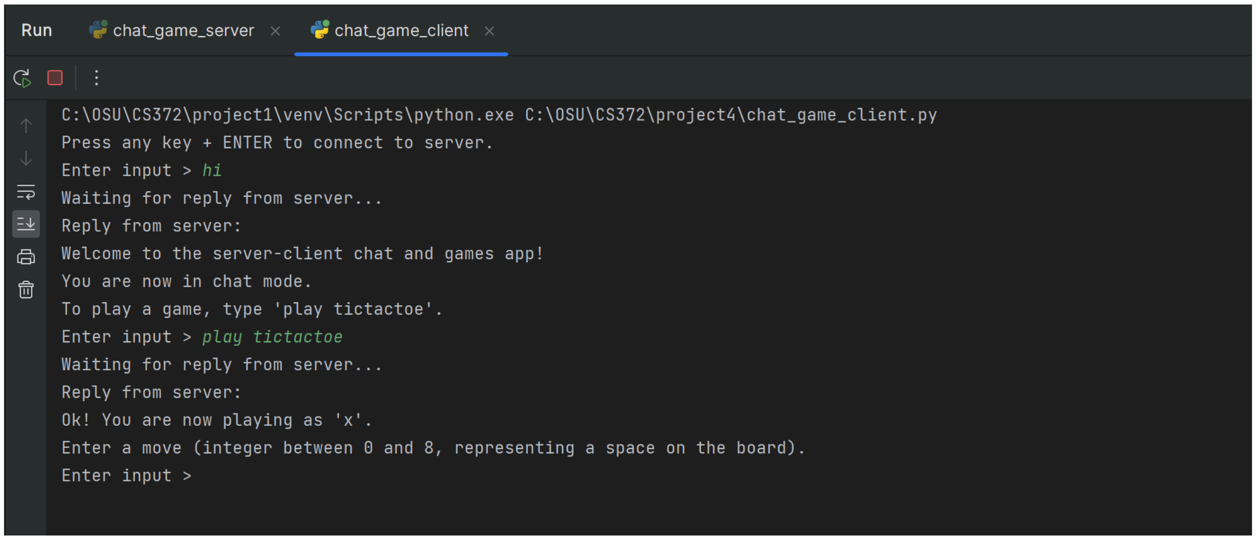


```
C:\OSU\CS372\project1\venv\Scripts\python.exe C:\OSU\CS372\project4\chat_game_server.py
Server ready. Waiting for client...
Client has connected! Waiting for next message...
Message from client: hi
Enter input > hi back
Waiting for reply from client...
Message from client: /q
Client has closed the connection with command /q. Shutting down!

Process finished with exit code 0
```



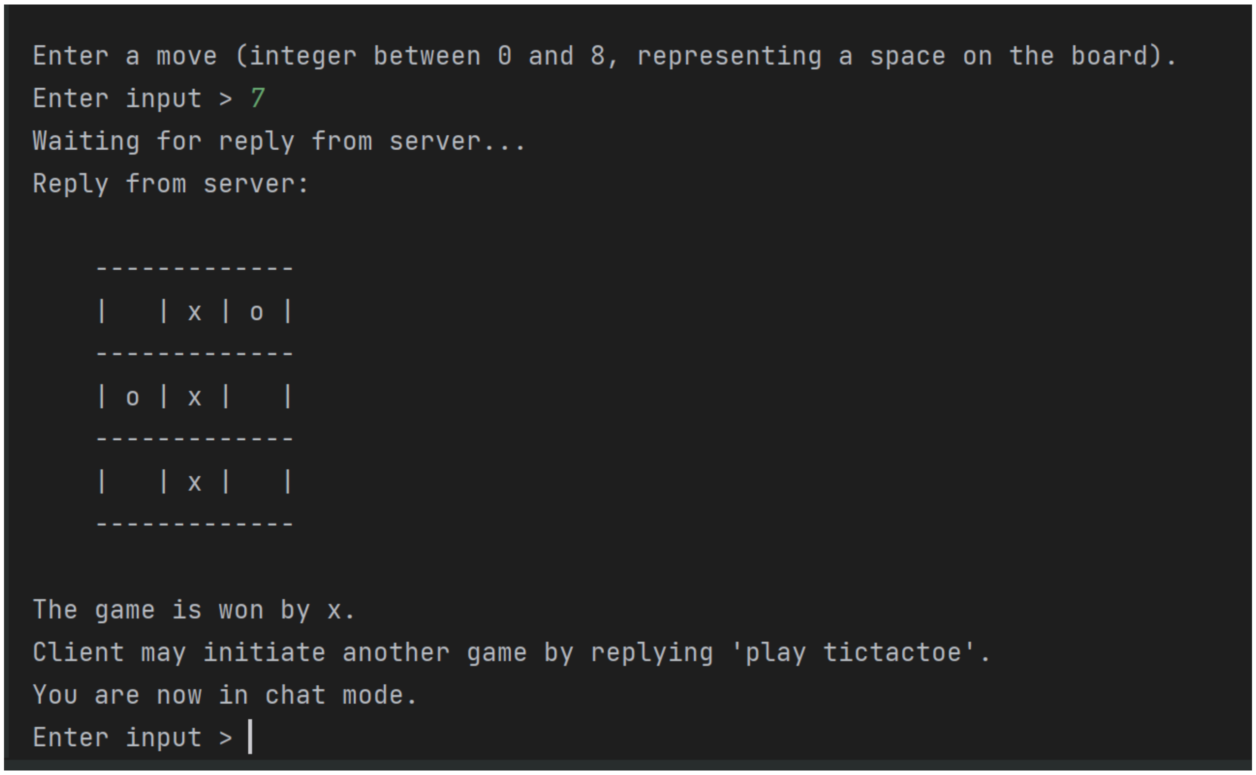
```
Cannot send empty message!
Enter input > hi
Waiting for reply from server...
Reply from server:
hi back
Enter input > /q
Waiting for reply from server...
Reply from server:
Client has closed the connection with command /q. Shutting down!
Enter input >
Cannot send empty message!
Enter input > f
Waiting for reply from server...
Traceback (most recent call last):
  File "C:\OSU\CS372\project4\chat_game_client.py", line 25, in <module>
    message = sock.recv(4096).decode()
              ^^^^^^^^^^^^^^^
ConnectionAbortedError: [WinError 10053] An established connection was aborted by the software in your host machine

Process finished with exit code 1
```

3. Initiating a game of tic-tac-toe:

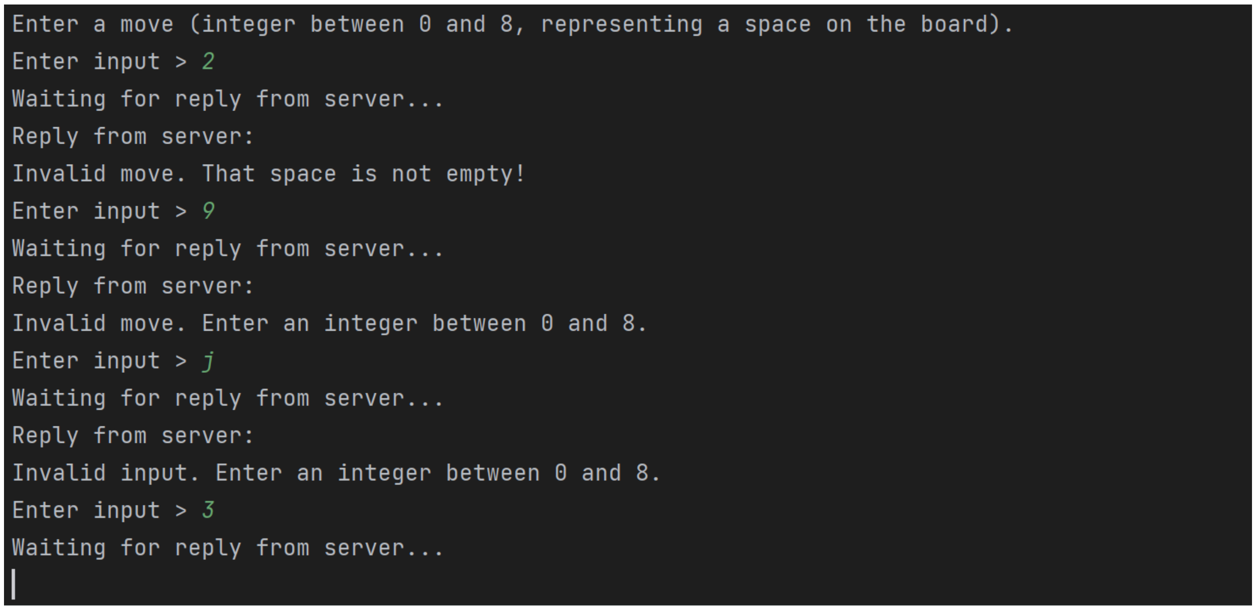


```
C:\OSU\CS372\project1\venv\Scripts\python.exe C:\OSU\CS372\project4\chat_game_client.py
Press any key + ENTER to connect to server.
Enter input > hi
Waiting for reply from server...
Reply from server:
Welcome to the server-client chat and games app!
You are now in chat mode.
To play a game, type 'play tictactoe'.
Enter input > play tictactoe
Waiting for reply from server...
Reply from server:
Ok! You are now playing as 'x'.
Enter a move (integer between 0 and 8, representing a space on the board).
Enter input >
```

4. Completing a game of tic-tac-toe and returning to chat mode:


```
Enter a move (integer between 0 and 8, representing a space on the board).
Enter input > 7
Waiting for reply from server...
Reply from server:

    -------------
    |   | x | o |
    -------------
    | o | x |   |
    -------------
    |   | x |   |
    -------------

The game is won by x.
Client may initiate another game by replying 'play tictactoe'.
You are now in chat mode.
Enter input >
```

5. Error handling for bad input:


```
Enter a move (integer between 0 and 8, representing a space on the board).
Enter input > 2
Waiting for reply from server...
Reply from server:
Invalid move. That space is not empty!
Enter input > 9
Waiting for reply from server...
Reply from server:
Invalid move. Enter an integer between 0 and 8.
Enter input > j
Waiting for reply from server...
Reply from server:
Invalid input. Enter an integer between 0 and 8.
Enter input > 3
Waiting for reply from server...
```

COMMENTS/QUESTIONS

This is all my own work. I built on my knowledge of socket connections from previous projects, but the code is substantially different from other classwork.

Implementing the game itself was simple, as there are only 8 win conditions for tic tac toe. All of the game logic is handled on the server side, which updates and clears the board as necessary, sends over “pretty printed” boards so the client can keep track of the game, checks for win conditions, and toggles between chat mode and tictactoe mode. The server also handles all of the error checking.

The client code only checks that input isn’t empty before sending and doesn’t “know” about anything to do with the tictactoe game, so to speak.

The only issue that I encountered with this project was a hanging recv after the first back-and-forth between client and server. I turned to Ed for this and found quickly that I just needed to accept the connection outside of my infinite send/recv loop.