```
#########################################################
#     Tic-Tac-Toe Client                                #
#     a client to connect to the ttt server.            #
#                                                       #
#     Noah Jaffe                                        #
#     TCP Socket Programming                            #
#     CMSC 481                                          #
#     11/05/2018                                        #
#########################################################


###################################
# Usage: python tttc.py [-c] [-s serverIP]
# client starts a TCP connection to the server with the given IP
address and default port number: 13037
# client port number should be dynamically allocated
# client must be able to handle at least these 2 command line options
#     -s serverIP
#           Server - required - specifies the IP address of the
server.
#     -c
#           Client Start - the client will send the first move.
#           If the '-c' option is not used, the AI makes the first
move.
# NOTES:
# server stores the current game state
# server must keep the board correctly, not overwriting moves and
knowing when a win has occurred
# It can play as stupidly as you like.
#
# SUBMIT:
#     Working documented code.
#           For partial credit, you must be able to handle one client
at a time.
#           For full credit, you must handle multiple clients at a
time.
#     Protocol Specification documenting the messages that are sent
between the client
#           and the server which would allow someone to develop their
own client or
#           server to interact with yours.
###################################
from socket import *
from socket import error as socket_err
import getopt, sys, struct

#CONSTANTS & GLOBALS
client_socket = socket(AF_INET, SOCK_STREAM)
```

```
TTT_SERVER_PORT = 13037
TTT_PRTCL_REQUEST_FIRST_ARGS = "Please send an unsigned int
representing if the client wishes to make the first move.\n\t0 --
sever should go first\n\t1 -- client should go first"
TTT_PRTCL_GOT_FIRST_ARGS_ERR = "Failed to receive proper game
initiation arguments. Terminating connection.\nNext time " +
TTT_PRTCL_REQUEST_FIRST_ARGS
TTT_PRTCL_INSTRUCTIONS = "Welcome to Tic Tac Toe!\nEnter [0-8] for the
position of your move, or 9 to quit:\n0|1|2\n-----\n3|4|5\n-----\n6|7|
8\n"
TTT_PRTCL_INVALID_CLIENT_INPUT = "Invalid input, try again."
TTT_PRTCL_REQUEST_CLIENT_TURN = " | | \n-----\n | | \n-----\n | |
\nEnter [0-8] for the position of your move, or 9 to quit:\n"
TTT_PRTCL_CLIENT_ERR = "Sorry, that was invalid input. Please try
again."
TTT_PRTCL_TERMINATE = 0
TTT_PRTCL_EXPECTING_NO_RESPONSE = 1
TTT_PRTCL_EXPECTING_INT_RESPONSE = 2
TTT_PRTCL_EXPECTING_FIRST_ARGS_RESPONSE = 3
TTT_PRTCL_PACKED_UNSIGNED_INT_SIZE = 4 #4 is the size of a packed '!I'
value


def recv_server_response():
      '''
      RECIEVING FROM SERVER TO CLIENT:
            RECV PACKED: MESSAGE RESPONSE LENGTH
            unpack '!I' and .recv that many bytes
            RECV: MESSAGE
            RECV PACKED: EXPECTING RESPONSE VAL
            unpack '!i' and add to ret_list
      Receives the size of the next incoming response, and the next
incoming response.

      RETURNS:
            server_response -- the response received.
            [<EXPECTING RESPONSE>, <MESSAGE>]
            <EXPECTING RESPONSE> Valid values are:
                  TTT_PRTCL_TERMINATE
                        -- connection will be closing, message is the
last message from the server.
                  TTT_PRTCL_EXPECTING_NO_RESPONSE
                        -- expecting no response, message is just a
message.
                  TTT_PRTCL_EXPECTING_INT_RESPONSE
                        -- expecting a single digit integer response,
message is a prompt for the user
                  TTT_PRTCL_EXPECTING_FIRST_ARGS_RESPONSE
```

```
                        -- expecting the TTT_PRTCL_REQUEST_FIRST_ARGS
response, message is instructions.
            <MESSAGE> Valid values are:
                  A string with a message.
      '''
      ret_list = []
      #recv a message length from the server
      server_msg_len_buf = recvall(TTT_PRTCL_PACKED_UNSIGNED_INT_SIZE)
      if not server_msg_len_buf:
            return None
      server_msg_len, = struct.unpack("!I", server_msg_len_buf)

      #recv a message from the server
      server_msg = recvall(server_msg_len).decode()
      ret_list.append(server_msg)

      #recv an int value of the expexted response value
      expecting_response_buf =
recvall(TTT_PRTCL_PACKED_UNSIGNED_INT_SIZE)
      if not expecting_response_buf:
            return None
      expecting_response, = struct.unpack("!I",
expecting_response_buf)
      #add expcted response value to list
      try:
            ret_list.insert(0, expecting_response)
      except:
            #if insertion failed bc of some reason or expecting
response is None, then default to termination
            ret_list.insert(0, TTT_PRTCL_TERMINATE)

      return ret_list

def recvall(length):
      '''
      Receives messages of a specific size (size) from the connection
(conn)

      ARGUMENTS:
            conn -- the connection
            size -- number of bytes to read

      RETURNS:
            <bytes> -- the (packed/encoded) message from the client
            None -- if connection failed before reading in the message
      '''
      encoded_msg = b''
      while length:
```

```
            temp = client_socket.recv(length)
            if not temp:
                  return None
            encoded_msg += temp
            length -= len(temp)

      return encoded_msg


def parse_cmd_line_args(argv):
      '''
      returns an unsigned int value representing if the client has
first move or not

      argv -- list with [-c] [-s serverIP]

      RETURNS:
            <CLIENT FIRST>
            <CLIENT FIRST> Valid values are:
                  0 -- server has first move
                  1 -- client has first move

      '''
      cmd_line_args = 0 #default is the server starts
      if "-c" in argv:
            cmd_line_args = 1
      '''
      #parse arguments
      for i, v in enumerate(argv):
            if "-c" in v:
                  #overwrite cmd_line_args so that client starts
                  cmd_line_args = 1
      '''
      return cmd_line_args


def get_single_digit_response(message):
      '''
      Propmpts user with message, and gets a single digit from user
input.
      RETURNS:
            <unsigned int> -- a single digit.
      '''
      user_input = "default_invalid"

      #validate user input
      while not(user_input.isdigit()):
            #prompt user with message
```

```
            user_input = input(message)
      try:
            return int(user_input[0])
      except:
            print("ERROR @ TTTC.py::get_single_digit_response():
FAILED TO INTERPRET USER INPUT... TRYING AGAIN")
            return get_single_digit_response(message)

def send_single_digit_response(num):
      '''
      Sends an unsigned int value to the server

      SENDING FROM CLIENT TO SERVER:
            pack single digit val '!I'
            SEND PACKED: SINGLE DIGIT VAL

      '''
      client_socket.send(struct.pack('!I', num))


def play_game(argv):
      '''
      LOOPS UNTIL GAME IS DONE OR THE CLIENT QUITS
      recv a message from the server. The message can be any of the
following:
            a message with the active game board and instructions for
the client user.
            an end of game message.
            an error message.
      '''
      #get next server response
      server_response = recv_server_response()
      while server_response:

            #check if termination message
            if server_response[0] == TTT_PRTCL_TERMINATE:
                  #print the server message
                  print(server_response[1])
                  print("Connection terminating, goodbye.")
                  client_socket.close()   #TODO: IS THIS VALID?
                  return True

            elif server_response[0] ==
TTT_PRTCL_EXPECTING_NO_RESPONSE:
                  #print the server message
                  print(server_response[1])

            elif server_response[0] ==
```

```
TTT_PRTCL_EXPECTING_INT_RESPONSE:
                  #send single digit integer response and pass the
message prompt to the getter
                  num = get_single_digit_response(server_response[1])
                  send_single_digit_response(num)

            elif server_response[0] ==
TTT_PRTCL_EXPECTING_FIRST_ARGS_RESPONSE:
                  #send if the client goes first
                  num = parse_cmd_line_args(argv)
                  send_single_digit_response(num)
                  print("successfully set
TTT_PRTCL_REQUEST_FIRST_ARGS")

            #get next server response
            server_response = recv_server_response()

def main(argv):
      '''
      Connect to server and starts the game.

      argv -- list with [-c] [-s serverIP]
      '''
      ttt_server_name = 'ERROR'
      #set server
      for i, v in enumerate(argv):
            if v == "-s":
                  ttt_server_name = argv[i + 1]

      #attempt to connect to server
      try:
            client_socket.connect((ttt_server_name, TTT_SERVER_PORT))
      except socket_err:
            #clean exit
            print("ERROR @ TTTC.py::main(): FAILED TO CONNECT TO TCP
SERVER... ABORTING")
            sys.exit(1)

      #play the game
      play_game(argv)

      client_socket.close()
      sys.exit(0)

if __name__ == '__main__':
      '''
      Call main() with the appropriately parsed command line argument
list
```

```
'''
# read commandline arguments
fullCmdArguments = sys.argv

#get the argument parts
argumentList = fullCmdArguments[1:]

#send it to main
main(argumentList)
```