2023학년도 1학기
정보통신공학

Term Project:
Tic-Tac-Toe Socket Implementation Technical Report

과 목 명	정보통신공학
담 당 교 수	이형준
학 과	컴퓨터공학과
학 번	2071035
이 름	이소민 (LeeSomin)
제 출 일	2023.06.05

1. Code

1) ETTTP_TicTacToe.py

```
'''
  2071035 Lee Somin
  ETTTP_TicTacToe.py

  34743-02 Information Communications
  Term Project on Implementation of Ewah Tic-Tac-Toe Protocol

  Skeleton Code Prepared by JeiHee Cho
  May 24, 2023

 '''

import random
import tkinter as tk
from socket import *
import _thread

SIZE=1024

class TTT(tk.Tk):
    def __init__(self, target_socket,src_addr,dst_addr, client=True):
        super().__init__()

        self.my_turn = -1

        self.geometry('500x800')

        self.active = 'GAME ACTIVE'
        self.socket = target_socket

        self.send_ip = dst_addr
        self.recv_ip = src_addr

        self.total_cells = 9
        self.line_size = 3


        # Set variables for Client and Server UI
        ############## updated ############################
        if client:
            self.myID = 1    #0: server, 1: client
            self.title('34743-02-Tic-Tac-Toe Client')
```

```
            self.user = {'value': self.line_size+1, 'bg': 'blue',
                         'win': 'Result: You Won!', 'text':'O','Name':"ME"}
            self.computer = {'value': 1, 'bg': 'orange',
                             'win': 'Result: You Lost!', 'text':'X','Name':"YOU"}
        else:
            self.myID = 0
            self.title('34743-02-Tic-Tac-Toe Server')
            self.user = {'value': 1, 'bg': 'orange',
                         'win': 'Result: You Won!', 'text':'X','Name':"ME"}
            self.computer = {'value': self.line_size+1, 'bg': 'blue',
                         'win': 'Result: You Lost!', 'text':'O','Name':"YOU"}
        ######################################################


        self.board_bg = 'white'
        self.all_lines = ((0, 1, 2), (3, 4, 5), (6, 7, 8),
                          (0, 3, 6), (1, 4, 7), (2, 5, 8),
                          (0, 4, 8), (2, 4, 6))

        self.create_control_frame()

    def create_control_frame(self):
        '''
        Make Quit button to quit game
        Click this button to exit game

        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.control_frame = tk.Frame()
        self.control_frame.pack(side=tk.TOP)

        self.b_quit = tk.Button(self.control_frame, text='Quit',
                                command=self.quit)
        self.b_quit.pack(side=tk.RIGHT)
        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    def create_status_frame(self):
        '''
        Status UI that shows "Hold" or "Ready"
        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.status_frame = tk.Frame()
        self.status_frame.pack(expand=True,anchor='w',padx=20)

        self.l_status_bullet =
```

```
tk.Label(self.status_frame,text='O',font=('Helevetica',25,'bold'),justify='left')
        self.l_status_bullet.pack(side=tk.LEFT,anchor='w')
        self.l_status = tk.Label(self.status_frame,font=('Helevetica',25,'bold'),justify='left')
        self.l_status.pack(side=tk.RIGHT,anchor='w')
        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    def create_result_frame(self):
        '''
        UI that shows Result
        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.result_frame = tk.Frame()
        self.result_frame.pack(expand=True,anchor='w',padx=20)

        self.l_result = tk.Label(self.result_frame,font=('Helevetica',25,'bold'),justify='left')
        self.l_result.pack(side=tk.BOTTOM,anchor='w')
        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    def create_debug_frame(self):
        '''
        Debug UI that gets input from the user
        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.debug_frame = tk.Frame()
        self.debug_frame.pack(expand=True)

        self.t_debug = tk.Text(self.debug_frame,height=2,width=50)
        self.t_debug.pack(side=tk.LEFT)
        self.b_debug = tk.Button(self.debug_frame,text="Send",command=self.send_debug)
        self.b_debug.pack(side=tk.RIGHT)
        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


    def create_board_frame(self):
        '''
        Tic-Tac-Toe Board UI
        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.board_frame = tk.Frame()
        self.board_frame.pack(expand=True)

        self.cell = [None] * self.total_cells
        self.setText=[None]*self.total_cells
        self.board = [0] * self.total_cells
```

```
            self.remaining_moves = list(range(self.total_cells))
            for i in range(self.total_cells):
                self.setText[i] = tk.StringVar()
                self.setText[i].set("  ")
                self.cell[i] = tk.Label(self.board_frame,
highlightthickness=1,borderwidth=5,relief='solid',
                                        width=5, height=3,
                                        bg=self.board_bg,compound="center",
                                        textvariable=self.setText[i],font=('Helevetica',30,'bold'))
                self.cell[i].bind('<Button-1>',
                                  lambda e, move=i: self.my_move(e, move))
                r, c = divmod(i, self.line_size)
                self.cell[i].grid(row=r, column=c,sticky="nsew")

            #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

        def play(self, start_user):
            '''
            Call this function to initiate the game

            start_user: if its 0, start by "server" and if its 1, start by "client"
            '''
            #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
            self.last_click = 0
            self.create_board_frame()
            self.create_status_frame()
            self.create_result_frame()
            self.create_debug_frame()
            self.state = self.active
            if start_user == self.myID:
                self.my_turn = 1
                self.user['text'] = 'X'
                self.computer['text'] = 'O'
                self.l_status_bullet.config(fg='green')
                self.l_status['text'] = ['Ready']
            else:
                self.my_turn = 0
                self.user['text'] = 'O'
                self.computer['text'] = 'X'
                self.l_status_bullet.config(fg='red')
                self.l_status['text'] = ['Hold']
                _thread.start_new_thread(self.get_move,())
            #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
    def quit(self):
        '''
        Call this function to close GUI
        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.destroy()
        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    def my_move(self, e, user_move):
        '''
        Read button when the player clicks the button

        e: event
        user_move: button number, from 0 to 8
        '''
        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv

        # When it is not my turn or the selected location is already taken, do nothing
        if self.board[user_move] != 0 or not self.my_turn:
            return
        # Send move to peer
        valid = self.send_move(user_move)

        # If ACK is not returned from the peer or it is not valid, exit game
        if not valid:
            self.quit()

        # Update Tic-Tac-Toe board based on user's selection
        self.update_board(self.user, user_move)

        # If the game is not over, change turn
        if self.state == self.active:
            self.my_turn = 0
            self.l_status_bullet.config(fg='red')
            self.l_status ['text'] = ['Hold']
            _thread.start_new_thread(self.get_move,())
        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

    def get_move(self):
        '''
        Function to get move from other peer
        Get message using socket, and check if it is valid
        If is valid, send ACK message
        If is not, close socket and quit
```

```
        '''
        ###################  Fill Out  ########################
        msg =  self.socket.recv(1024) # get message using socket
        msg = msg.decode()  # decode message

        # check if the message is in format of ETTTP Protocol
        msg_valid_check = check_msg(msg,self.recv_ip)

        if msg_valid_check: # Message is not valid
            # exit game
            self.socket.close()
            self.quit()
            return
        else:  # If message is valid - send ack, update board and change turn
            ackMsg = make_ack(msg)  # make ACK message
            self.socket.send(ackMsg.encode())   # send encoded ACK message to peer
            msg_split = msg.split() # get array of splitted message
            loc = int(msg_split[3][10])*3+int(msg_split[3][12]) # received next-move

        ##########################################################


            #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
            self.update_board(self.computer, loc, get=True)
            if self.state == self.active:
                self.my_turn = 1
                self.l_status_bullet.config(fg='green')
                self.l_status ['text'] = ['Ready']
            #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


    def send_debug(self):
        '''
        Function to send message to peer using input from the textbox
        Need to check if this turn is my turn or not
        '''

        if not self.my_turn:
            self.t_debug.delete(1.0,"end")
            return
        # get message from the input box
        d_msg = self.t_debug.get(1.0,"end")
        d_msg = d_msg.replace("\\r\\n","\r\n")   # msg is sanitized as \r\n is modified when
it is given as input
```

```
        self.t_debug.delete(1.0,"end")

        ###################  Fill Out  ######################
        # chek if debug message is in ETTTP format
        if check_msg(d_msg,self.send_ip):
            return
        # split debug message to get the user move position
        d_msg_split = d_msg.split()
        # convert coordinate position to array location
        user_move = int(d_msg_split[3][10])*3+int(d_msg_split[3][12])
        # When it is not my turn or the selected location is already taken, do nothing
        if self.board[user_move] != 0:
            return

        # Send move to peer
        self.socket.send(d_msg.encode())

        # If ACK is not returned from the peer or it is not valid, exit game
        ackMsg = self.socket.recv(1024)
        ackMsg = ackMsg.decode()
        # if ACK message is not proper, exit game
        if(check_msg(ackMsg,self.recv_ip) or check_ack(d_msg,ackMsg)):
            self.quit()

        loc = user_move # peer's move, from 0 to 8

        ##########################################################

        #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
        self.update_board(self.user, loc)

        if self.state == self.active:      # always after my move
            self.my_turn = 0
            self.l_status_bullet.config(fg='red')
            self.l_status ['text'] = ['Hold']
            _thread.start_new_thread(self.get_move,())

        #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


    def send_move(self,selection):
        '''
        Function to send message to peer using button click
        selection indicates the selected button
```

```
        '''
        row,col = divmod(selection,3)
        ###################  Fill Out  ######################

        # send message and check ACK
        msg = 'SEND
ETTTP/1.0\r\nHost:'+str(self.send_ip)+'\r\nNew-Move:('+str(row)+','+str(col)+')\r\n\r\n'
        self.socket.send(msg.encode())
        # get ACK message
        ackMsg = self.socket.recv(1024)
        ackMsg = ackMsg.decode()     # decode ACK message
        # if ack message is not proper, this move is not valid
        if(check_msg(ackMsg,self.recv_ip) or check_ack(msg,ackMsg)):
            return False
        # if the function did not return, return true meaning this move is valid
        return True
        ##########################################################


    def check_result(self,winner,get=False):
        '''
        Function to check if the result between peers are same
        get: if it is false, it means this user is winner and need to report the result first
        '''
        ###################  Fill Out  ######################
        # set message informing the winner
        resultMsg = 'RESULT ETTTP/1.0\r\n
Host:'+str(self.send_ip)+'\r\nWinner:'+winner+'\r\n\r\n'
        recvMsg = ''
        ackMsg = ''

        if get: # if in situation of getting final move
            # recieve result message from peer
            recvMsg = self.socket.recv(1024)
            recvMsg = recvMsg.decode()  # decode message recieved
            # send ack if message is correct
            if check_msg(recvMsg,self.recv_ip):
                return False
            else:
                ackMsg = make_ack(recvMsg)  # generate ack message
                self.socket.send(ackMsg.encode())    # send ack message
                # send result of mine and wait for ack
                self.socket.send(resultMsg.encode())
                ackMsg = self.socket.recv(1024)
```

```
                ackMsg = ackMsg.decode()     # decode ack message
                # check if ack recieved is correct
                if(check_msg(ackMsg,self.recv_ip) or check_ack(resultMsg,ackMsg)):
                    return False

        else:    # if in situation of sending final move
            # send the result message to peer
            self.socket.send(resultMsg.encode())
            # wait for ack
            ackMsg = self.socket.recv(1024)
            ackMsg = ackMsg.decode()
            # check if ack is correct
            if(check_msg(ackMsg,self.recv_ip) or check_ack(resultMsg,ackMsg)):
                return False
            else:
                # get result message from peer
                recvMsg = self.socket.recv(1024)
                recvMsg = recvMsg.decode()
                # send ack if message is correct
                if check_msg(recvMsg,self.recv_ip):
                    return False
                else:
                    ackMsg = make_ack(recvMsg)  # generate ack message
                    self.socket.send(ackMsg.encode())    # send ack message
        # correct result message should indicate different winner('ME'<->'YOU') for each other
        recvWinner = recvMsg.split()
        if 'Winner:'+winner == recvWinner[3]:
            return False
        return True
        ##########################################################


    #vvvvvvvvvvvvvvvvvvv  DO NOT CHANGE  vvvvvvvvvvvvvvvvvvv
    def update_board(self, player, move, get=False):
        '''
        This function updates Board if is clicked

        '''
        self.board[move] = player['value']
        self.remaining_moves.remove(move)
        self.cell[self.last_click]['bg'] = self.board_bg
        self.last_click = move
        self.setText[move].set(player['text'])
        self.cell[move]['bg'] = player['bg']
```

```
            self.update_status(player,get=get)

    def update_status(self, player,get=False):
        '''
        This function checks status - define if the game is over or not
        '''
        winner_sum = self.line_size * player['value']
        for line in self.all_lines:
            if sum(self.board[i] for i in line) == winner_sum:
                self.l_status_bullet.config(fg='red')
                self.l_status ['text'] = ['Hold']
                self.highlight_winning_line(player, line)
                correct = self.check_result(player['Name'],get=get)
                if correct:
                    self.state = player['win']
                    self.l_result['text'] = player['win']
                else:
                    self.l_result['text'] = "Somethings wrong..."

    def highlight_winning_line(self, player, line):
        '''
        This function highlights the winning line
        '''
        for i in line:
            self.cell[i]['bg'] = 'red'

    #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

# End of Root class

def check_msg(msg, recv_ip):
    '''
    Function that checks if received message is ETTTP format
    '''
    ####################  Fill Out  ########################
    # split message to check if it is in ETTTP format(starting with "ETTTP/1.0"
    # and to check if the destination of the message equals to reciever ip
    msgSplit = msg.split()
    if(msgSplit[1]== 'ETTTP/1.0' and msgSplit[2] == 'Host:'+str(recv_ip)):
        # return False if there is no problem
        return False
    # return True if there is any problem
    return True
    ########################################################
```

```
def check_ack(msg,ackmsg):
    '''
    Function that checks if recieved ack message is proper
    '''
    # split messages and check if the number of elements are same
    # and if the ack message starts with word "ACK"
    msgSplit = msg.split()
    ackmsgSplit = ackmsg.split()
    # return True if there is any problem
    if(len(msgSplit)!=len(ackmsgSplit) or ackmsgSplit[0]!='ACK'):
        return True
    # check if the message except 'SEND' and 'ACK' is same
    for i in range(1,len(msgSplit)):
        if(msgSplit[i]!=ackmsgSplit[i]):
            return True
    # return Flase if there is no problem
    return False


def make_ack(msg):
    '''
    Function that generates the ack message of given message
    '''
    # split message and substitute the first word 'SEND' with 'ACK'
    msg_split = msg.split()
    ackMsg = 'ACK '
    for i in range(1,len(msg_split)):
        ackMsg+=msg_split[i]+'\r\n'
    ackMsg+='\r\n'
    return ackMsg
```

2) ETTTP_Server.py

```
'''
  ETTTP_Sever_skeleton.py

  34743-02 Information Communications
  Term Project on Implementation of Ewah Tic-Tac-Toe Protocol

  Skeleton Code Prepared by JeiHee Cho
  May 24, 2023

 '''
```

```
import random
import tkinter as tk
from socket import *
import _thread

from ETTTP_TicTacToe import TTT, check_msg, check_ack, make_ack


if __name__ == '__main__':

    global send_header, recv_header
    SERVER_PORT = 12000
    SIZE = 1024
    server_socket = socket(AF_INET,SOCK_STREAM)
    server_socket.bind(('',SERVER_PORT))
    server_socket.listen()
    MY_IP = '127.0.0.1'

    while True:
        client_socket, client_addr = server_socket.accept()

        start = random.randrange(0,2)    # select random to start

        ######################################################################
        # Send start move information to peer
        CLIENT_IP = client_socket.recv(1024)     #recive IP address of client from client
        CLIENT_IP = CLIENT_IP.decode()  # decode the IP message
        first_move = 'ME' if start == 0 else 'YOU' # if random number is 0, server goes first,
and if number is 1, client goes first
        startMsg = 'SEND ETTTP/1.0\r\nHost:'+str(CLIENT_IP)+'\r\nFirst-Move:'+ first_move +
'\r\n\r\n'
        client_socket.send(startMsg.encode())

        ########################### Fill Out ###################################
        # Receive ack - if ack is correct, start game
        ackMsg = client_socket.recv(1024)
        ackMsg = ackMsg.decode()
        # if message and ack is not valid, exit game
        if(check_msg(ackMsg,MY_IP) or check_ack(startMsg,ackMsg)):
            client_socket.close()
            break
        ######################################################################
```

```
            root = TTT(client=False,target_socket=client_socket,
src_addr=MY_IP,dst_addr=client_addr[0])
            root.play(start_user=start)
            root.mainloop()

            client_socket.close()

            break
    server_socket.close()
```

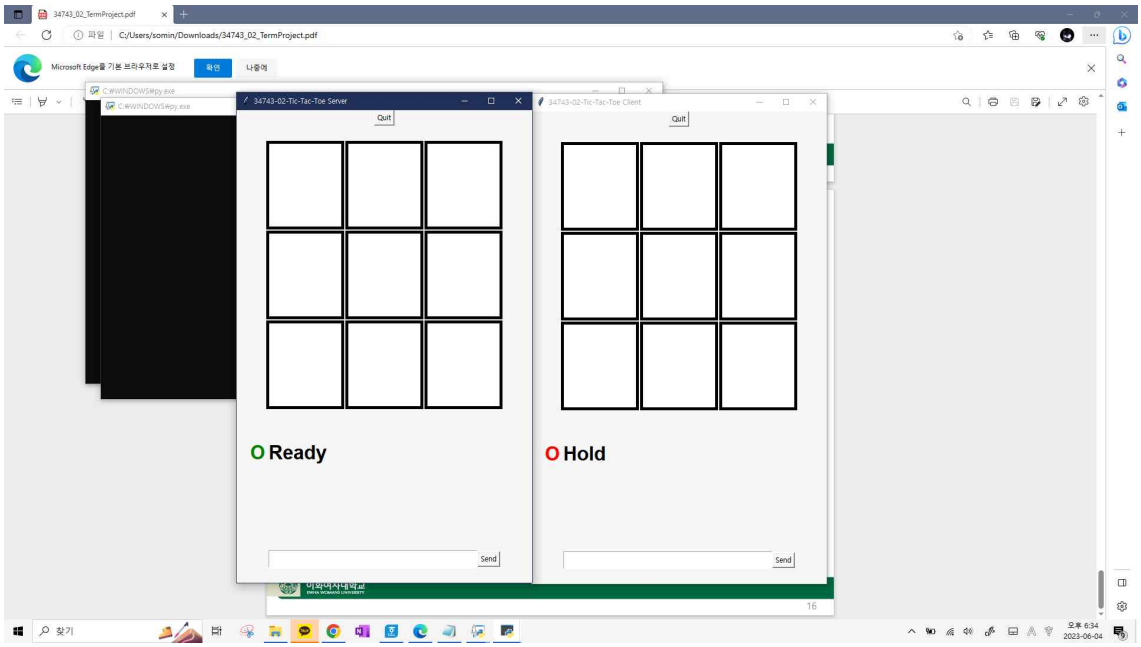
3) ETTTP_Client

```
'''
  ETTTP_Client_skeleton.py

  34743-02 Information Communications
  Term Project on Implementation of Ewah Tic-Tac-Toe Protocol

  Skeleton Code Prepared by JeiHee Cho
  May 24, 2023

 '''

import random
import tkinter as tk
from socket import *
import _thread

from ETTTP_TicTacToe import TTT, check_msg, check_ack, make_ack



if __name__ == '__main__':

    SERVER_IP = '127.0.0.1'
    MY_IP = '127.0.0.1'
    SERVER_PORT = 12000
    SIZE = 1024
    SERVER_ADDR = (SERVER_IP, SERVER_PORT)


    with socket(AF_INET, SOCK_STREAM) as client_socket:
        client_socket.connect(SERVER_ADDR)

        ######################################################################
```

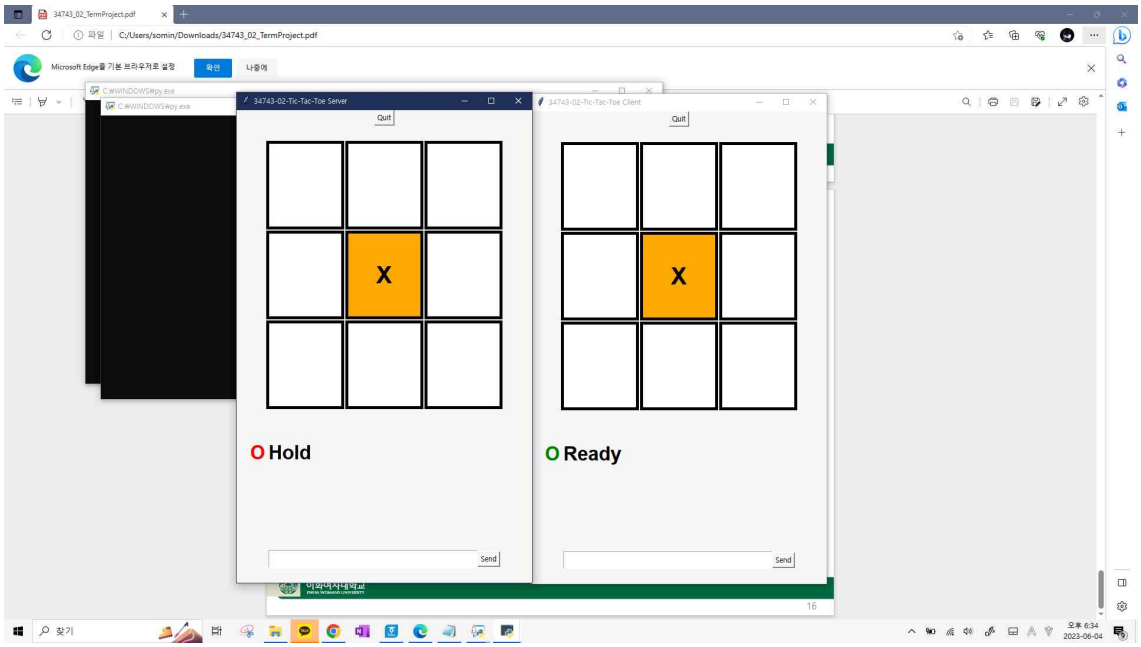
```
        # Receive who will start first from the server
        # send IP of self to server
        client_socket.send(MY_IP.encode())
        # recieve information about starting user
        startMsg = client_socket.recv(1024)
        startMsg = startMsg.decode()
        # check if the start message is proper
        if check_msg(startMsg,MY_IP):
            client_socket.close()
        # split message and check who is the first player
        startMsgSplit = startMsg.split()
        start = -1
        if startMsgSplit[3] == 'First-Move:ME':
            start = 0
        elif startMsgSplit[3] == 'First-Move:YOU':
            start = 1
        else:
            client_socket.close()

        ########################## Fill Out ###################################
        # Send ACK
        ackMsg = make_ack(startMsg)
        client_socket.send(ackMsg.encode())

        ######################################################################

        # Start game
        root = TTT(target_socket=client_socket, src_addr=MY_IP,dst_addr=SERVER_IP)
        root.play(start_user=start)
        root.mainloop()
        client_socket.close()
```

2. Result

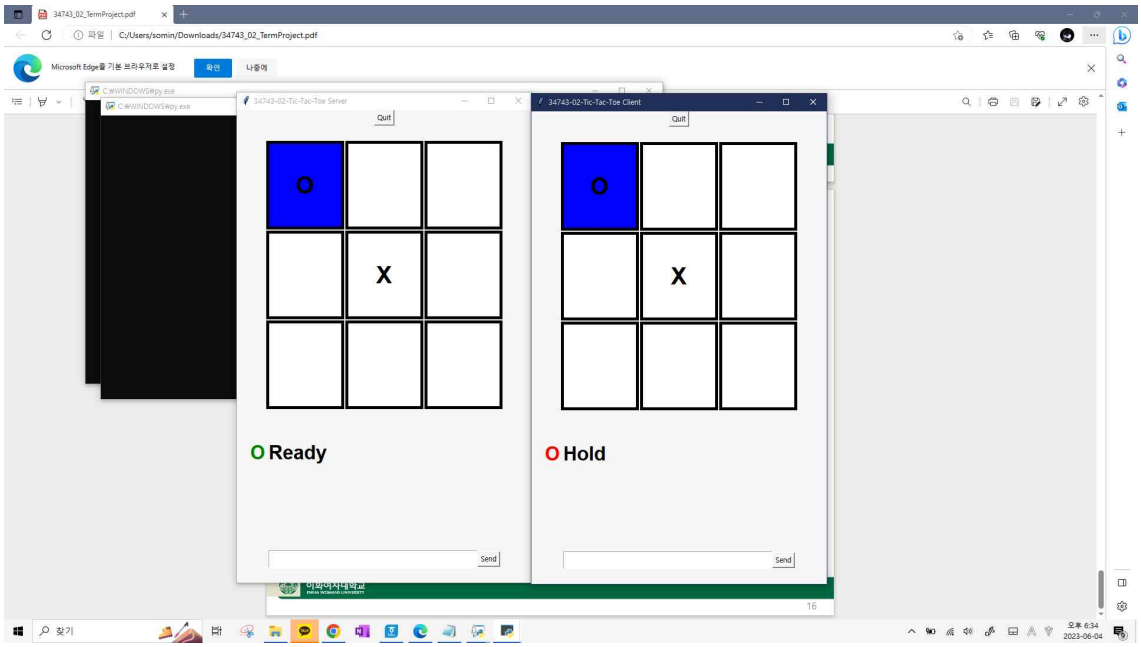
1) 처음 ETTTP_Server.py와 ETTTP_Client.py를 실행시켰을 때. (Server의 선공)



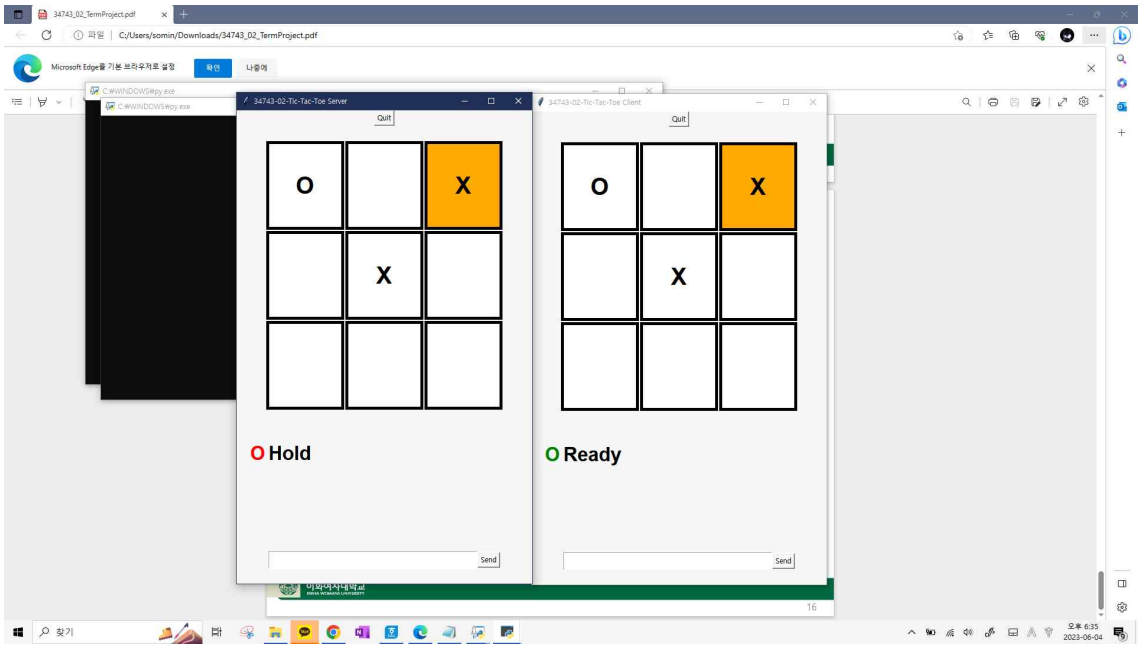
2) Server의 first move



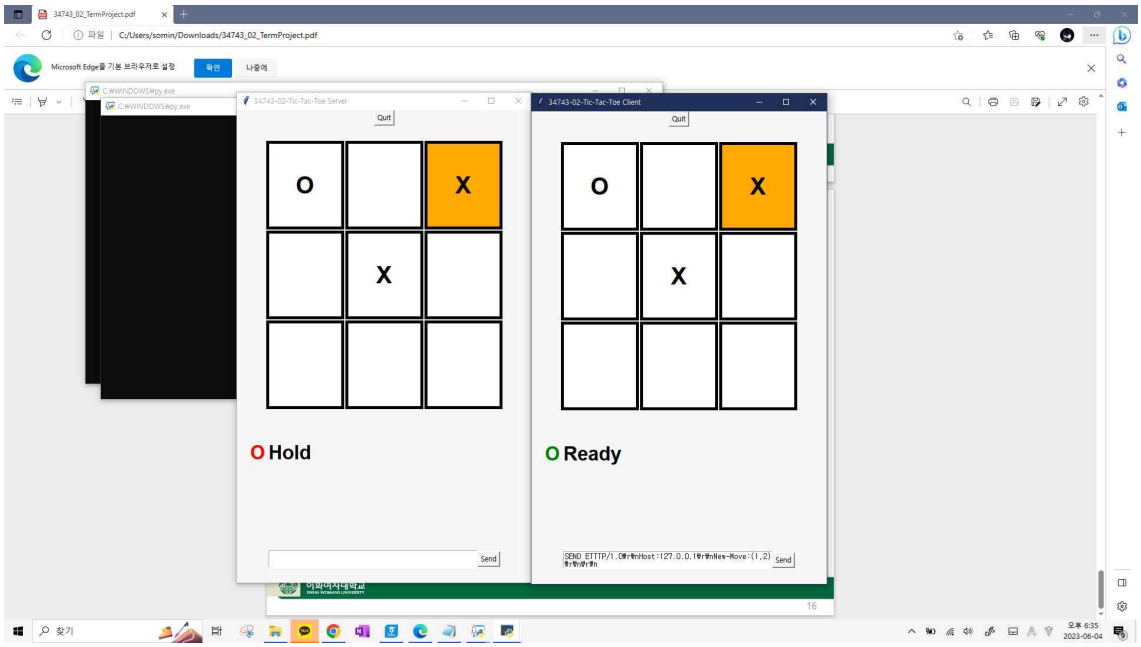
3) Client의 first move

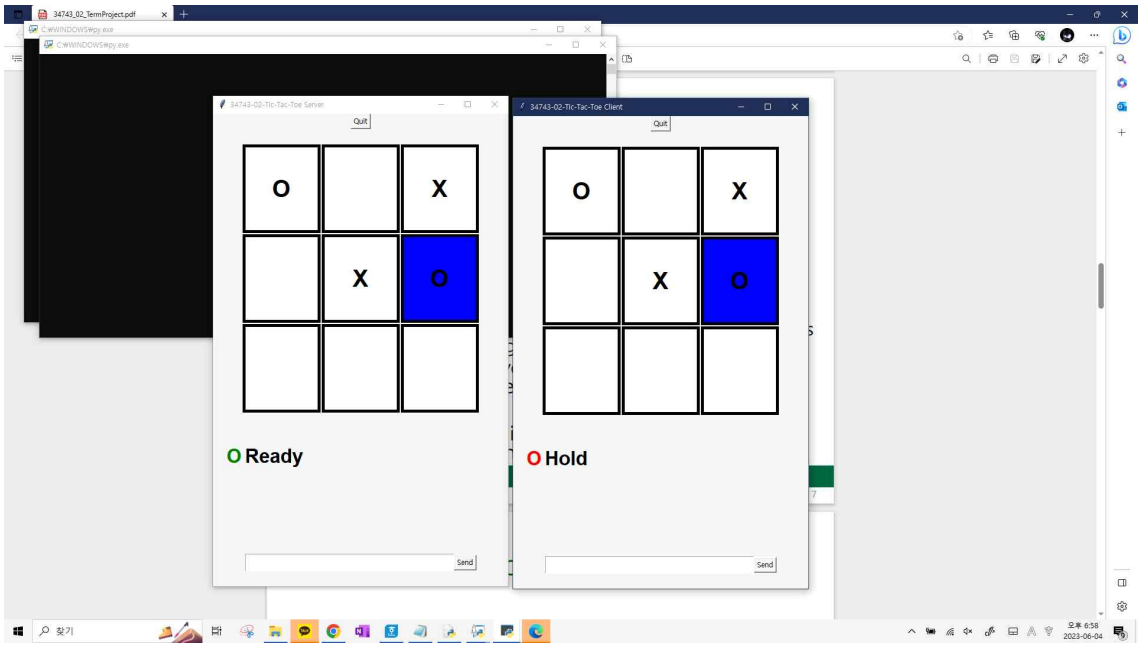


4) Server의 second move

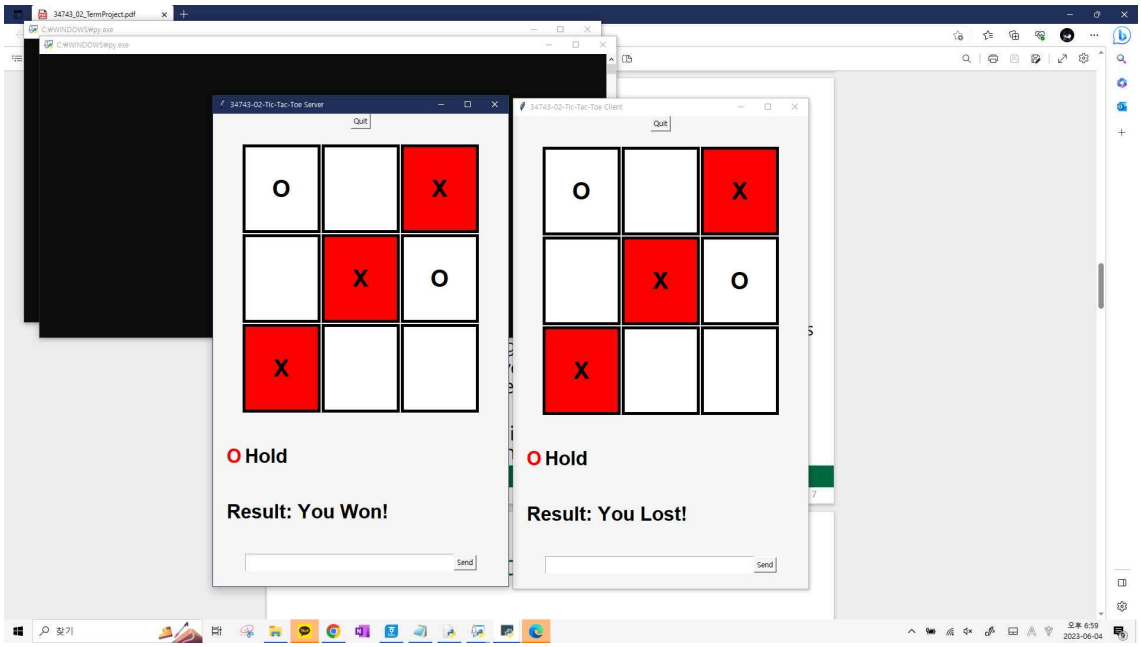


5) Client의 second move (debug message 이용)





6) Server의 third move (Server: 승리, Client: 패배)



3. Code Analysis

*Skeleton Code에서 주어진 부분을 제외한 새로 작성한 부분을 설명한다.

*모든 TCP send()는 특별한 언급 없이도 encode() 함수로 변환된 메시지를 전송하며 수신된 메시지는 별도의 언급 없이도 decode()함수로 원본메시지가 복원되어있음을 전제한다.

1) ETTTP_TicTacToe.py

TTT.get_move(self):

input: TTT self

return: -

이 함수는 peer가 놓은 수의 정보를 받아 자신의 board에 표시한다.

먼저, socket.recv() 함수를 통해 peer로부터 메시지를 받는다. TCP 메시지의 decoding 과정을 거친 후, check_msg() 함수를 통해 수신한 메시지가 ETTTP의 형식이 맞는지 확인하고, 유효한 형식이 아니면 소켓을 닫고 게임을 종료한다. 유효한 형식이면 make_ack()함수를 이용해 받은 메시지를 토대로 ACK 메시지를 생성하여 보낸다. acknowledge의 과정이 끝나면, split()함수로 수신한 메시지를 분해하여 peer가 선택한 위치를 좌표형식에서 배열의 index 형식으로 변환하여 변수 loc에 저장한다.

이때, 함수가 받는 메시지의 형식과 split으로 분리했을 때의 형식은 다음과 같다.

msg = 'SEND ETTTP/1.0\r\nHost:127.0.0.1\r\nNew-Move:(1,2)\r\n\r\n'

msg_split = ['SEND', 'ETTTP/1.0', 'Host:127.0.0.1', 'New-Move:(1,2)']

msg_split 배열의 3번원소는 'New-Move(r,c)'의 형식인데, 문자열의 10번 원소는 좌표의 행을, 12번 원소는 좌표의 열을 의미한다. 이때, 주어진 board의 행과 열은 모두 한자리수이므로 좌표는 반드시 10번과 12번을 행과 열로 가진다. Tic-Tac-Toe 게임의 game board는 왼편과 같은 번호를 가지는데, 이는 (행좌표*3+열좌표)와 같다. 따라서 msg_split[3]을 이용해 int(msg_split[3][10])*3+int(msg_split[3][12])과 같이 board에 표시할 위치의 번호인 변수 loc을 지정해준다. 이후, update_board 함수를 호출하여 수신한 위치정보를 자신의 보드에 갱신하고 my_turn 변수와 GUI 요소들의 상태를

Tic-Tac-Toe board		
0	1	2
3	4	5
6	7	8

변경해 자신의 순서로 넘어온다.

TTT.send_debug(self):
input: TTT self
return: -
이 함수는 debug message textbox를 통해 입력된 문자열이 ETTTP의 형태가 맞는지 확인 후 peer에게 전송하고 자신의 board를 갱신한다.
자신의 순서가 아니면 debug textbox의 내용을 지우고 함수를 반환한다. 반환되지 않았다면 textbox의 내용을 가져와 이스케이프 문자가 중복되는 현상을 수정하고 다음을 수행한다. 이 함수는 디버그 메시지가 ETTTP의 형식인지 확인한 다음, split() 함수로 디버그 메시지를 분해하여 이번 순서에서 선택한 위치를 좌표형식에서 배열의 index 형식으로 변환하여 변수 user_move에 저장한다. 형식 변환은 앞서 기술한 get_move() 함수에서 수신한 메시지에서 위치 좌표를 변환하는 방법과 동일하다. 만약, 변환된 위치의 board[user_move]의 값이 0이 아니어서 이미 O나 X가 존재하면, 아무 동작도 하지 않은 채 함수를 반환한다. 해당 위치가 비어있다면, socket.send() 함수를 사용해 디버그 메시지를 전송하고 ack를 기다린다. ack가 수신되면 ack메시지가 ETTTP 형식인지, 송신한 디버그 메시지와 내용이 일치하는지 check_msg() 함수와 check_ack() 함수를 통해 알아내고 만약 오류가 있다면 게임을 종료한다. loc은 이번 순서에서 선택된 board의 위치인 user_move와 같으며 이를 이용하여 자신의 game board를 갱신한다.

TTT.send_move(self,selection):
input: TTT self, int selection
return: boolean
이 함수는 board에서 클릭으로 선택된 위치를 peer에게 전송한다.
divmod() 함수를 통해 받은 selection 위치를 행,열의 형태로 바꾸어 row, col에 저장한다. send_ip와 row, col을 조합하여 전송할 메시지를 생성한다. 생성된 메시지를 socket.send()를 통해 송신하고 ack 메시지를 기다린다. ack 메시지가 수신되면 check_msg()와 check_ack()를 통해 ack 메시지가 올바른지 확인하고 해당 순서에서 둔 수가 유효하면 True를, 유효하지 않으면 False를 반환한다.

TTT.check_result(self, winner, get = False):
input: TTT self, string winner, boolean get
return: boolean
이 함수는 게임의 결과를 peer에게 전송하고 게임의 결과가 유효한지 확인하여 boolean형을 반환한다. 만약 자신이 마지막 수를 두었다면 결과 메시지를 전송하고 올바른 ack을 받은 후 peer가 연산한 결과 메시지를 받고 ack을 보낸 후 게임의 결과가 유효한지 확인하며, 자신이 마지막 수의 정보를 peer에게서 받는다면 peer에게서 결과를 먼저 수신하고 ack을 전송한 후 자신의 결과 메시지를 peer에게 전달하여 ack을 수신하여 게임 결과의 유효성을 점검한다. 게임의 결과가 유효하면 True값을 반환하고, 유효하지 않으면 False를 반환한다.
게임의 유효성을 확인할 때 자신이 보낸 결과 메시지와 peer로부터 받은 결과 메시지의 Winner 값을 비교하여 그 값이 같으면 오류를 검출한다. Server가 게임에서 이겼다고 가정하자. 이때, Server와 Client의 Result Message는 다음과 같다.

Server: RESULT ETTTP/1.0\r\n Host:ClientIP+\r\nWinner:“ME”\r\n\r\n
Client: RESULT ETTTP/1.0\r\n Host:ServerIP\r\nWinner:“YOU”\r\n\r\n

이렇듯, Server의 입장에서는 자신이 이겼으며, Client의 입장에서는 peer가 이겼으므로 Winner가 ME와 YOU로 갈린다. 따라서 받은 결과 메시지와 보낸 결과 메시지의 Winner값이 상이하면 유효한 게임

결과이다.

check_msg(msg, recv_ip):
input: string msg, string recv_ip
return: boolean
이 함수는 메시지가 유효한 ETTTP 형식인지 확인한다.
처음으로 split()함수를 통해 msg를 분해하고 2번 원소가 'ETTTP/1.0'의 형식인지 확인한다. 또한, 3번 원소의 Host IP가 수신자의 IP와 같으면 유효한 ETTTP 메시지이다.
이 함수는 msg가 유효한 ETTTP이면 False를 반환하고 유효하지 않으면 True를 반환한다.

check_ack(msg,ackmsg):
input: string msg, string ackmsg
return: boolean
이 함수는 ACK 메시지가 유효한지 검사한다.
받은 ackmsg와 msg를 split()함수로 분해한 후 ACK 메시지가 'ACK'로 시작하는지의 여부와 분해된 문자열 배열의 길이를 비교하여 한 조건이라도 만족하지 못하면 유효하지 않은 ACK로 처리한다. 앞의 조건을 만족하며 첫 원소 ('SEND'와 'ACK')를 제외한 모든 원소가 서로 일치하면 유효한 ACK 메시지이다.
이 함수는 ACK 메시지가 유효하면 False를 반환하고 유효하지 않으면 True를 반환한다.

make_ack(msg):
input: string msg
return: string ackMsg
이 함수는 받은 msg를 바탕으로 ACK 메시지를 생성하여 반환한다.
받은 msg 문자열을 split()함수로 분해한 후 'ACK '에 0번 원소를 제외한 msg_split의 원소를 이어붙이고 끝에 '\r\n'을 더해 ACK메시지를 생성한다. 모든 원소가 더해졌으면 '\r\n'을 한번 더 추가하여 메시지의 끝을 알린다. 마지막으로, 이 함수는 생성된 문자열 ackMsg를 ACK 메시지로서 반환한다.

2) ETTTP_Server.py
* ETTTP TCP통신에 ETTTP_TicTackToe.py 코드의 check_ack()와 make_ack() 함수가 필요하므로 import한다.
line36 - line43:
이 범위의 코드는 client로부터 peer의 IP주소를 수신하여 목적지IP로 사용하며 선공 사용자가 누구인지 Client에 알린다. 다항연산자를 사용해 무작위로 뽑은 start 변수가 0이면 'ME'를, 1이면 'YOU'를 startMsg에 조합해 소켓을 통해 encoding된 메시지를 전송한다.

line44 - line52:
이 범위의 코드는 보낸 First-Move 메시지에 대한 ACK를 기다리며, ACK를 수신하면 check_msg() 함수와 check_ack() 함수를 통해 유효성을 확인하고 ACK가 유효하지 않으면 소켓을 닫고 종료한다. ACK가 유효하면 게임을 시작한다.

3) ETTTP_Client.py
* ETTTP TCP통신에 ETTTP_TicTackToe.py 코드의 check_ack()와 make_ack() 함수가 필요하므로 import한다.

line34 - line53:

이 범위의 코드는 Server로 자신의 IP주소를 보내 Server가 목적지로 사용할 IP를 알리고 선공 사용자가 누구인지 Server로부터 메시지를 수신한다. check_msg()함수로 검사해 메시지가 유효하지 않으면 소켓을 닫고 종료한다. 메시지가 유효한 ETTTP 형식이면 startMsg를 split()함수로 분해하여 First-Move가 ME이면 Server를 의미하는 0을, YOU이면 Client를 의미하는 1을 start변수에 저장한다. 선공 사용자가 ME와 YOU 이외의 값을 가지면 소켓을 닫고 종료한다.

Client가 수신하게 되는 First-Move 메시지의 형식과 분해한 startMsgSplit배열은 다음과 같다.

startMsg: SEND ETTTP/1.0\r\nHost:CLIENT_IP\r\nFirst-Move:FirstPlayer\r\n\r\n

startMsgSplit = ['SEND', 'ETTTP/1.0', 'Host:CLIENT_IP', 'First-Move:FirstPlayer']

따라서 선공 사용자에 대한 정보는 startMsgSplit배열의 3번 원소에 저장된다.

line54 - line57:

이 범위의 코드는 make_ack()함수를 사용하여 수신한 First-Move 메시지에 대한 ACK를 전송한다. 위의 모든 과정에서 오류로 인해 소켓이 닫히지 않았다면 게임을 시작한다.