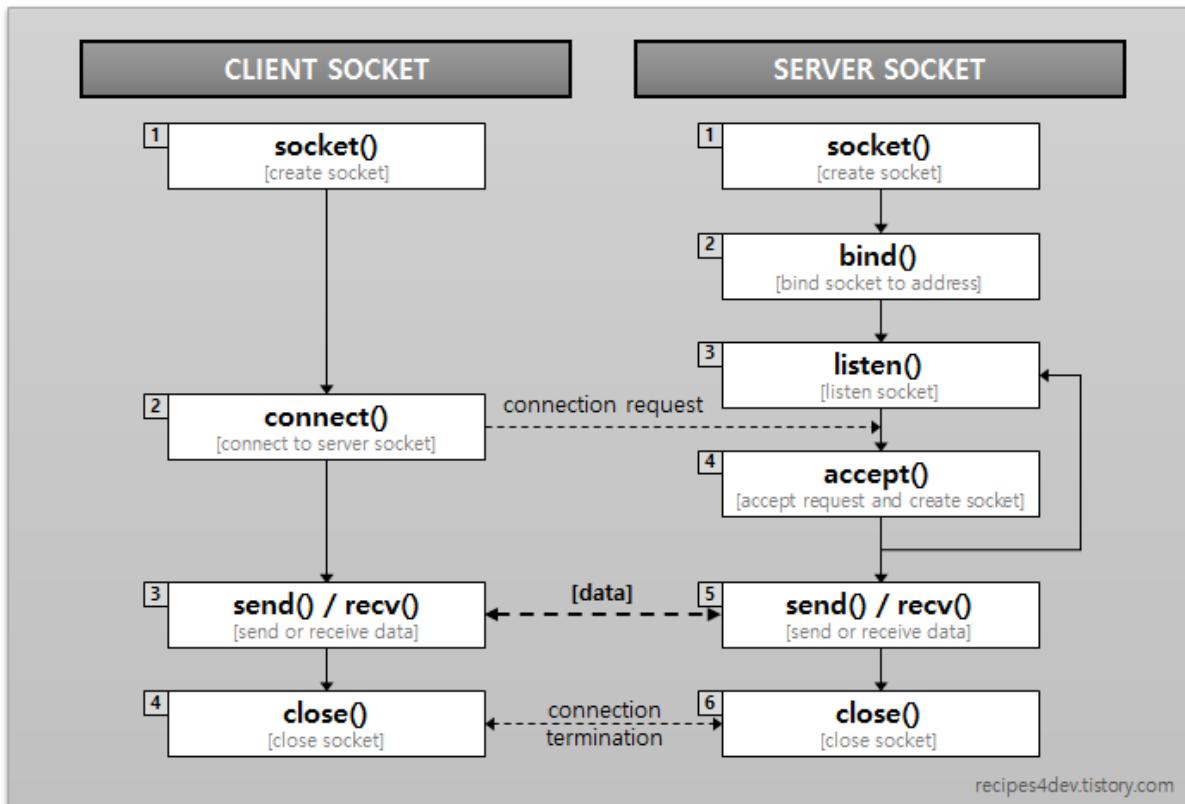


# ETTTP Project Report

## 1. Socket 통신에 대하여

소켓은 프로토콜, IP 주소, 포트 넘버로 정의됩니다. 소켓은 두 호스트를 연결해주는 인터페이스의 역할을 합니다. 즉, 두 소켓이 연결되면 서로 데이터를 주고 받을 수 있는 데이터 통로가 만들어 집니다. 이러한 소켓은 역할에 따라 서버 소켓, 클라이언트 소켓으로 구분된다.



저희는 서버 1개와 여러개의 클라이언트가 통신하는 시스템이 아니라 서버 1개와 클라이언트 1개가 1대1로 통신하고 있는 상황이므로, peer to peer 통신이라고도 볼 수 있으나, 연결 시작을 요청하는 쪽을 client, 연결 요청을 받고 순서를 정해서 전달해주는 상대측을 server라고 하고 진행하였습니다. 기본적으로 소켓 통신에서 클라이언트와 서버의 흐름을 본다면, 서버는 `socket` 함수를 이용하여 소켓을 생성하고, `bind` 함수로 ip주소와 port 번호를 소켓과 묶습니다. 그리고 `listen()` 함수로 클라이언트의 요청을 계속 기다립니다. 클라이언트 측에서는 `socket()` 함수로 가장 먼저 소켓을 열고, `connect()` 함수를 이용하여 통신하고 싶은 서버로 연결 요청을 보냅니다. 서버는 `accept()` 함수로 client의 연결 요청을 받고 client와 통신할 소켓을 생성합니다. 이를 통해 클라이언트와 서버가 서로 메시지를 주고 받으며 통신을 진행하게 됩니다.

저희가 이전에 배웠던, HTTP통신과 Socket 통신을 비교해본다면, HTTP는 client가 요청을 보내는 경우에만 서버가 응답하는 simplex 방식이고, 서버의 응답을 받은 후에는 연결이 종료됩니다. 그러나 Socket 통신은 server와 client가 포트를 통해 실시간으로 full-duplex 통신을 진행합니다. 저희의 ETTTP 프로젝트는 tic-tac-toe 프로그램을 두 개의 서버를 열어서 실시간으로 양측에서 데이터를 주고받으며 게임을 진행해야 하기 때문에 socket 통신이 적절하였다고 볼 수 있습니다.

## 2. Code Analysis

**get\_move(self):** 상대방의 move 정보를 get하는 함수입니다. 소켓으로부터 받은 msg를 decode하여 check\_msg를 이용해 msg가 valid인지 확인합니다. msg가 valid하지 않을 경우 quit하고, valid할 경우엔 받은 msg를 parsing해서 필요한 정보(row, col)을 추출합니다. 이후 ACK 메시지를 소켓을 통해 보내고, row와 col 정보로 위치(loc)를 저장한 후 이를 이용해 보드에 값을 반영합니다.

**send\_debug(self):** move를 클릭하는 방법이 아닌 텍스트창으로 보내는 함수입니다. 자기 차례가 아닐 경우엔 텍스트창의 input을 discard하고, 입력받은 input을 format에 맞게 처리할 수 있도록 수정하는 과정을 거칩니다. 가공된 input은 row와 col 정보를 추출하기 위해 파싱(re.split을 이용)하였고, 추출된 row와 col 정보를 통해 loc의 값을 저장해 입력받은 move가 이미 선택된 위치인지 아닌지 판단할 때 사용되게 했습니다. 새로운 move가 선택된 위치이거나 send가 아닌 다른 메시지가 입력된 경우 return을 합니다. 자기 차례일 때 이전에 선택되지 않은 위치를 골랐다면 d\_msg를 상대방에게 전달하고, 이에 대한 ack\_msg(ACK 메시지)를 받아 decode합니다. check\_msg로 ack\_msg의 valid함을 확인하여 valid할 경우엔 자신의 board를 업데이트하는 과정을 거치고, 그렇지 않을 경우엔 return하며 종료됩니다.

**send\_move(self,selection):** selection의 위치의 button을 클릭해서 move를 보내는 함수입니다. selection 위치를 divmod를 이용해 row와 col으로 변환하고, row와 col 위치를 선택하겠다는 msg를 상대방의 소켓에 encoding하여 보냅니다. 상대방이 이에 대해 보낸 ACK(ack\_msg)를 받으면 이 ack\_msg의 valid함을 check\_msg로 판단한 후, valid함에 따라 False(이상 있음)나 True(이상 없음)를 return 합니다.

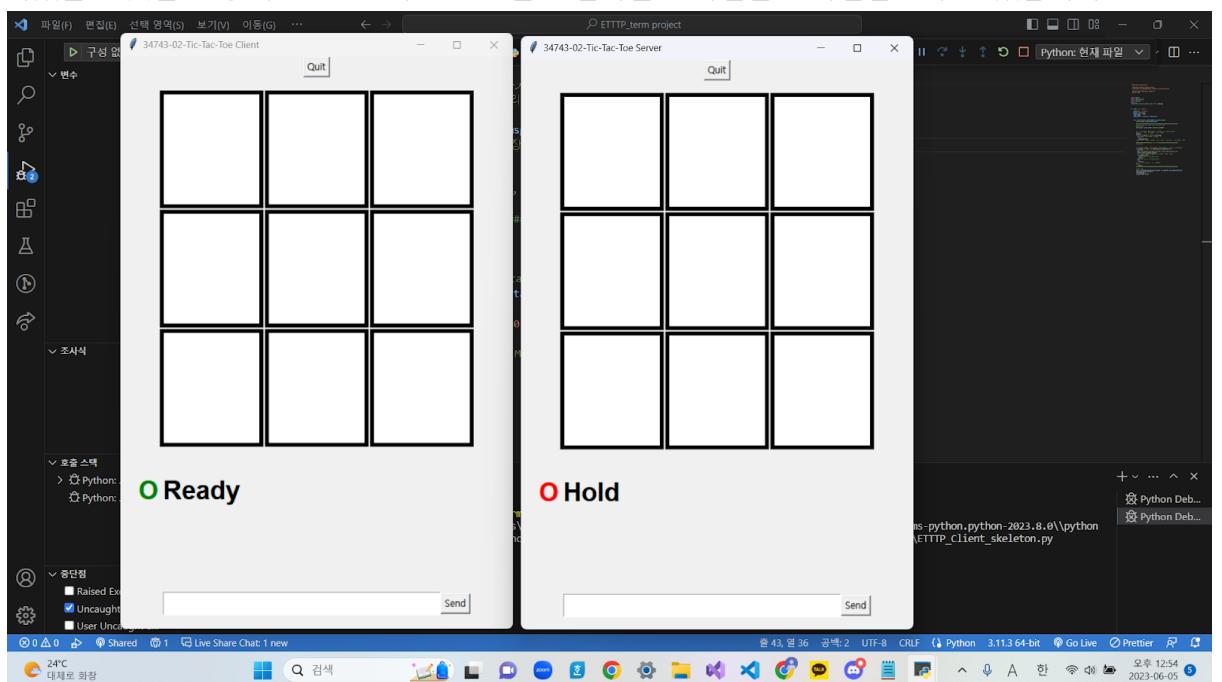
**check\_result(self,winner,get=False):** 이 함수는 update\_status 함수에서 보드의 값의 합이 winner\_sum인 경우 실행되는 함수입니다. 즉, 한 줄이 완성된 winner가 먼저 실행하는 함수로, winner는 자신이 이겼다는 msg를 상대에게 전송합니다. 진 상대방은 이긴 사람으로부터 받은 RESULT msg가 형식에 맞는지 확인하고, 자신의 winner 변수값이 "YOU"인지 확인합니다. "YOU"이면 상대방과 같은 결과를 나타낸 것이므로, 제대로 결과가 나온 것임을 알 수 있습니다. 그러나 형식에 맞지 않거나 winner 변수가 잘못된 경우 False를 리턴하여 something is wrong.. 이 화면에 출력되도록 합니다. 진 사람의 보드에서 "YOU"임이 확인 된 경우, 진 사람은 상대가 이겼다는 RESULT 메시지를 보냅니다. 이긴 사람은 그 메시지를 받고, valid한지 확인하고 True를 리턴합니다.

**check\_msg(msg,recv\_ip):** 해당 msg가 ETTTP 양식에 맞는지 판단하는 함수입니다. 전달받은 메시지의 각 부분들이 양식에 맞는지 확인하기 위해 re.split을 이용해 msg를 나눈 후, 이를 array에 저장하였습니다. array에는 ['SEND', 'ETTTP', '1.0', 'Host', '127.0.0.1', 'New-Move', '(2,3)]에는 이런 식으로 값이 저장되어 array[0]은 data type(send, ack, result 등), array[1]와 array[2]에는 각각 ETTTP와 version이 저장되며 array[3], array[4], array[5], array[6]에는 각각 Host, peer IP address, content(New-Move, Winner 등), 와 move 정보(row, column이나 YOU 등)가 저장됩니다. 먼저 array[0]이 SEND, ACK, RESULT 중에 해당하는, array[1]과 array[2]에 담긴 string이 정확히 "ETTTP"와 "1.0"인지를 비교하여 protocol과 version을 확인하였고, array[4]에 저장된 address가 recv\_ip가 맞는지 확인하도록 하였습니다. array[5]에 담긴 content가 First-Move/Winner일 때는 move 정보가 YOU 또는 ME여야하므로, 이것이 맞는지 확인하는 과정을 넣었고 content가 New-Move일 때는 move 정보에 담긴 row와 col 정보가 정수값이 맞는지 확인합니다. content가 형식에 맞지 않거나 제대로 된 move 정보가 아닐 경우 False를 return하도록 하였습니다. 위의 모든 과정 통과한 msg일 경우에만 True를 return할 수 있습니다.

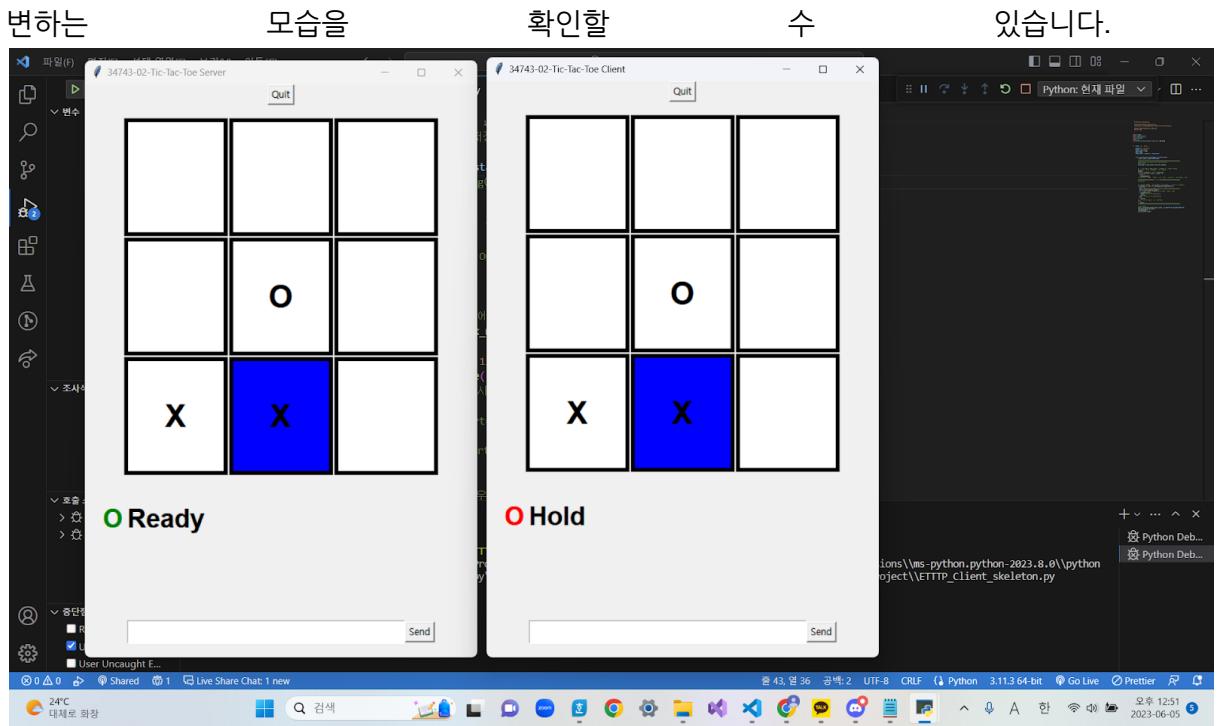
### 3. Test Case

#### <버튼을 클릭시 실행화면>

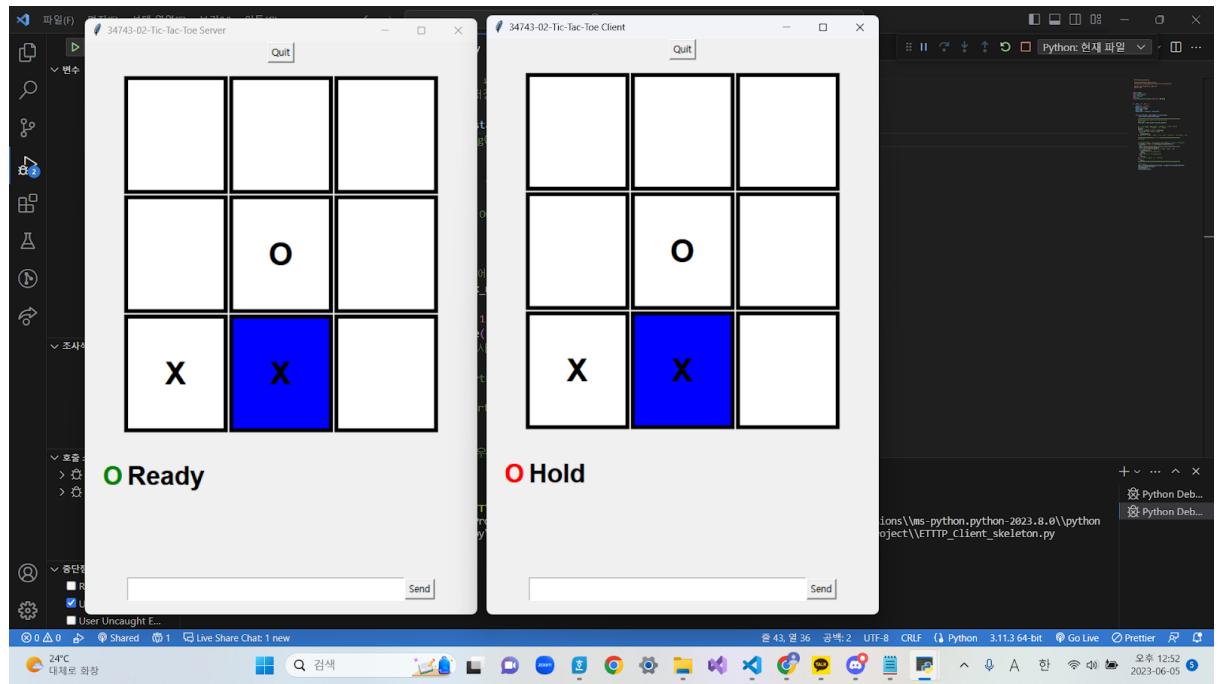
- 1) 시작 화면: 랜덤으로 누가 시작할지 결정되며, 현재 Client는 Ready, Server는 Hold가 떠있는 것을 통해 Client가 start를 한다는 사실을 확인할 수 있습니다.



- 2) 버튼을 누르며 플레이하는 화면입니다. 특정 위치의 버튼을 누르면 양 측의 화면에 모두 동일하게 업데이트가 되며, 해당 플레이어의 선택이 끝나면 각자의 Ready와 Hold가

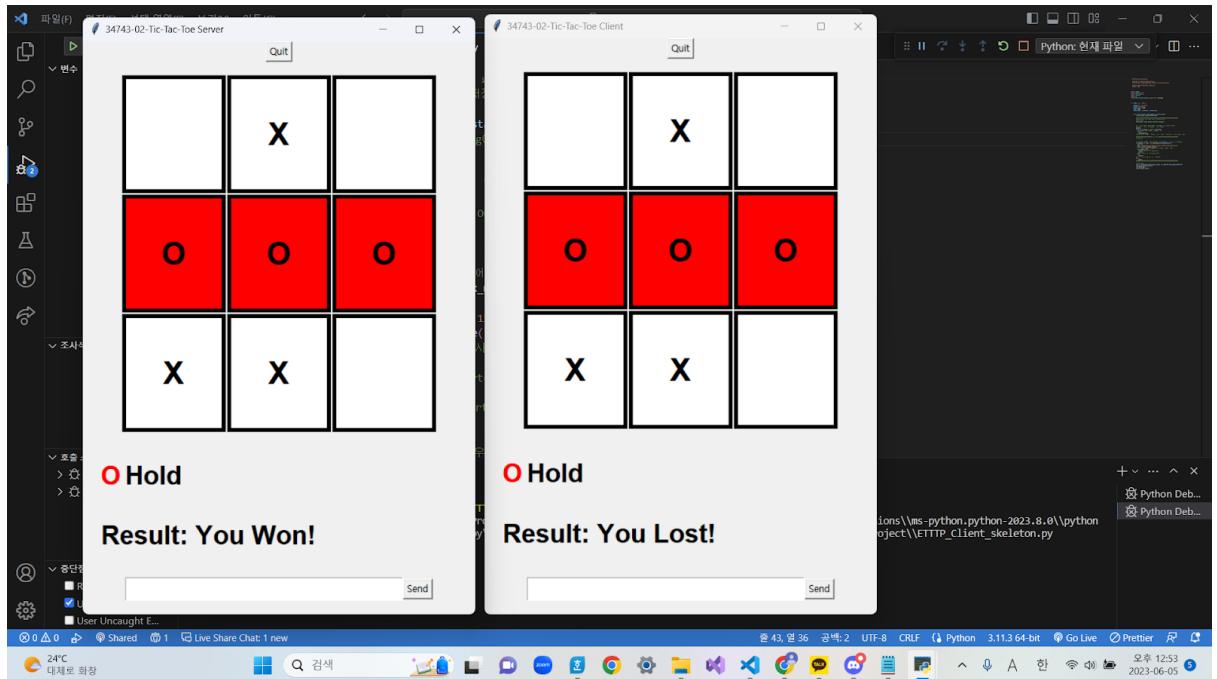


- 3) 자기 차례가 아닌 Client 창에서 버튼을 누르고, 자기 차례인 Server가 이미 눌려있는 위치(2,1)를 다시 눌렀음에도 화면이 2)와 달라지지 않은 것을 확인할 수 있습니다.



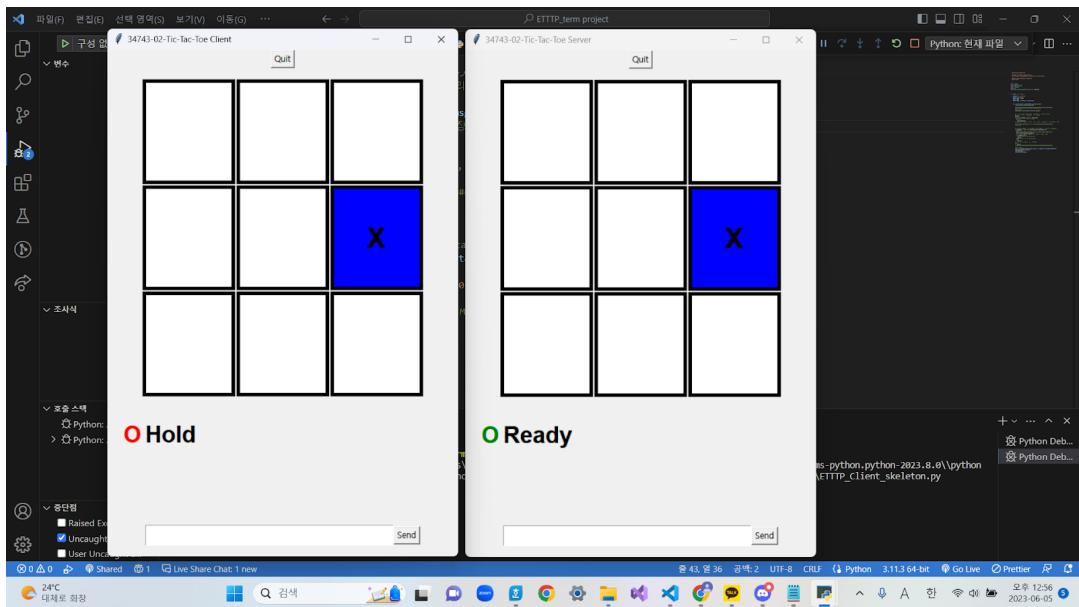
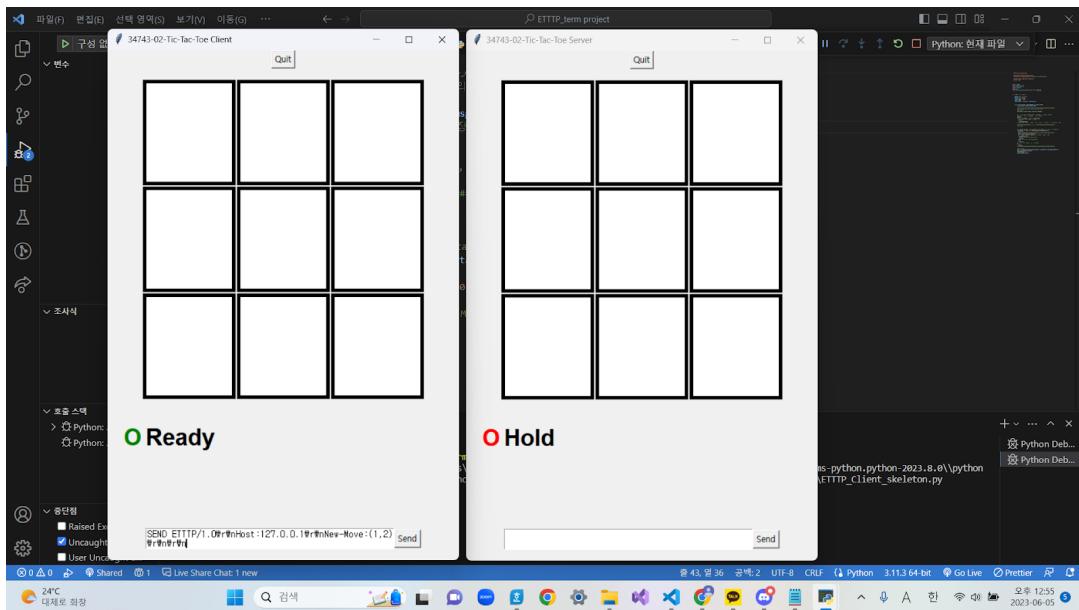
- 4) Server가 세 줄을 O로 만드면서 우승하였고, Server에게는 You Won!라는 Result 메시지가, Client에게는 You Lost!라는 Result 메시지가 출력됩니다. 이를 통해

check\_result가 제대로 작동함을 확인할 수 있습니다.

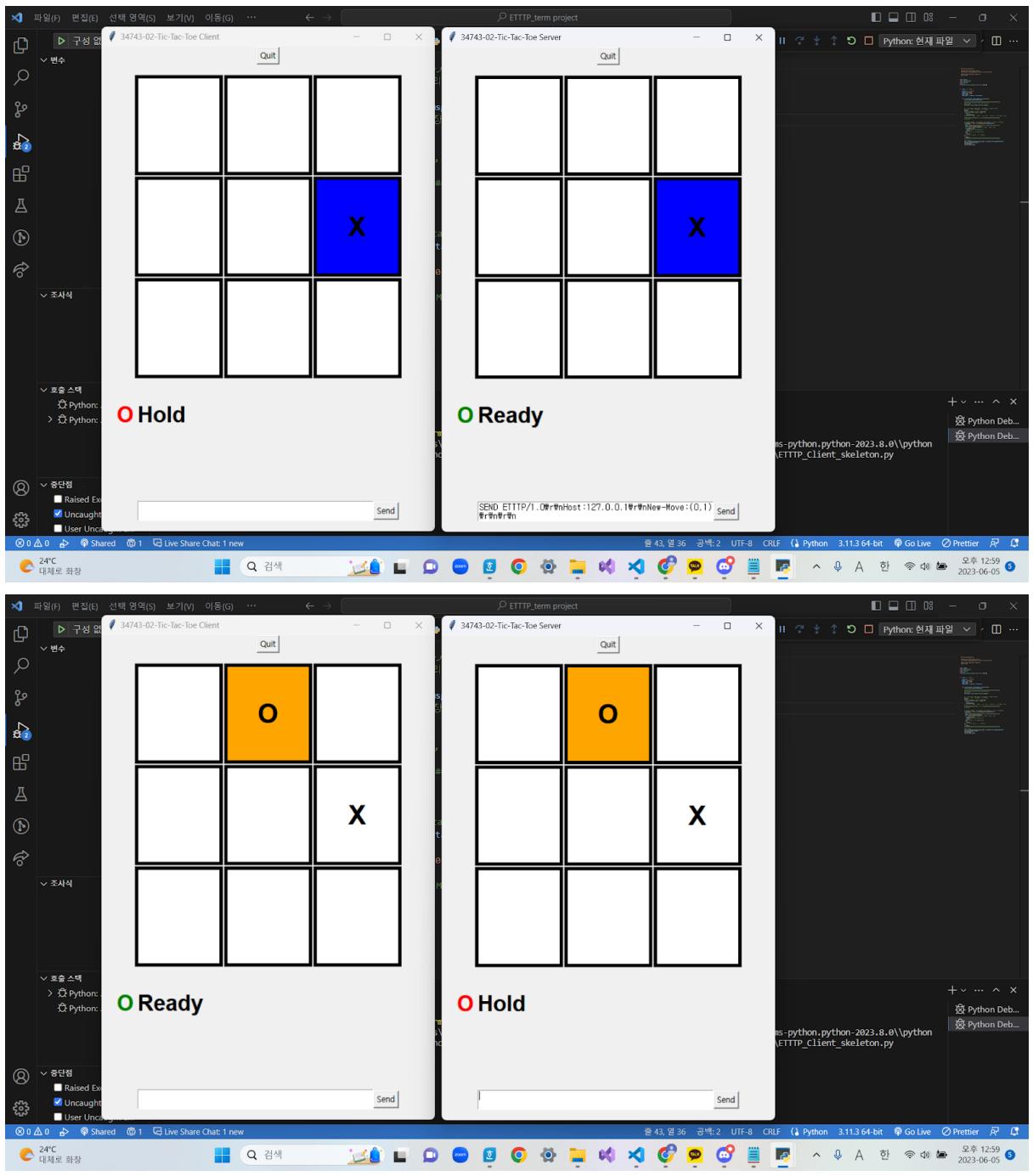


### <디버그 실행화면>

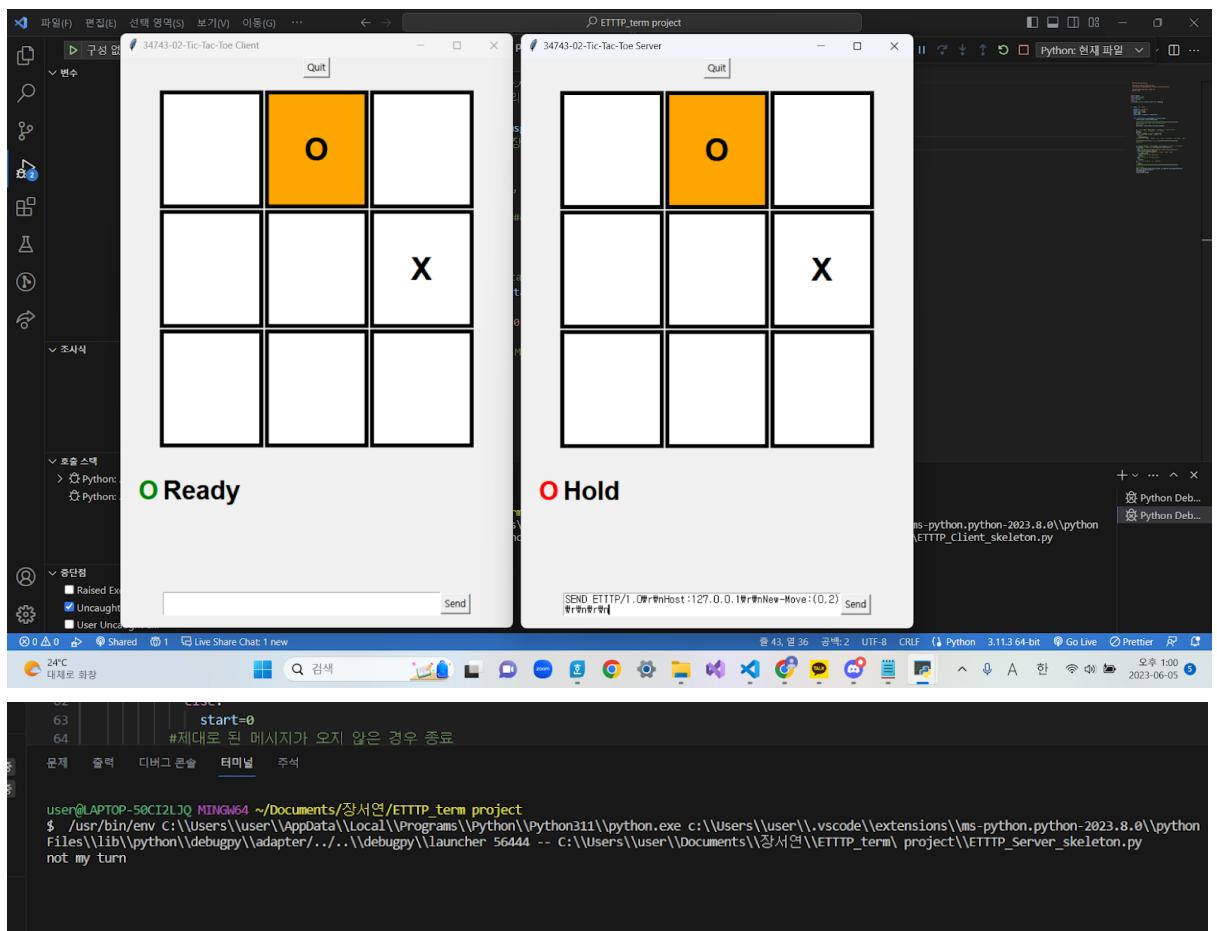
- 1) (1,2)로 New-Move를 하겠다는 정상적인 SEND 메시지를 보내보았습니다. ETTTP format에 맞는 메시지를 보냈을 때 자신과 상대방의 보드에 정상적으로 위치가 표시되는 것을 확인할 수 있습니다.



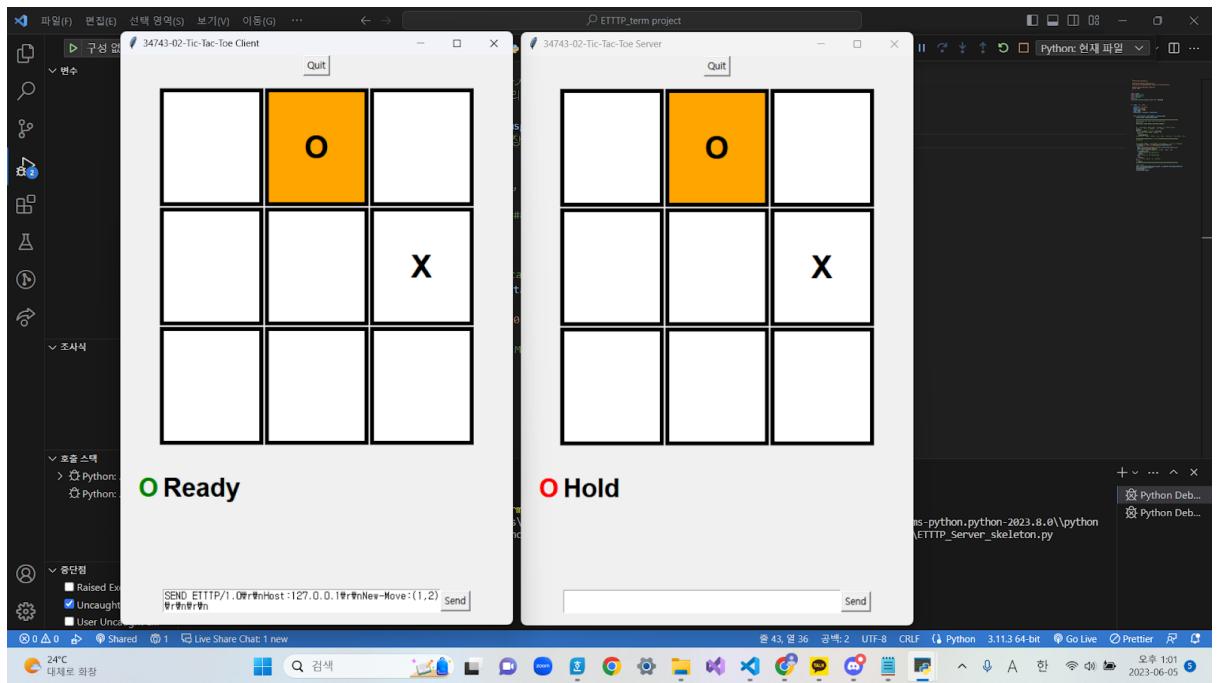
- 2) 상대측에서도 format에 맞는 정상적인 메시지(0,1)를 보냈을 때 보드에 정상적으로 표시됩니다.

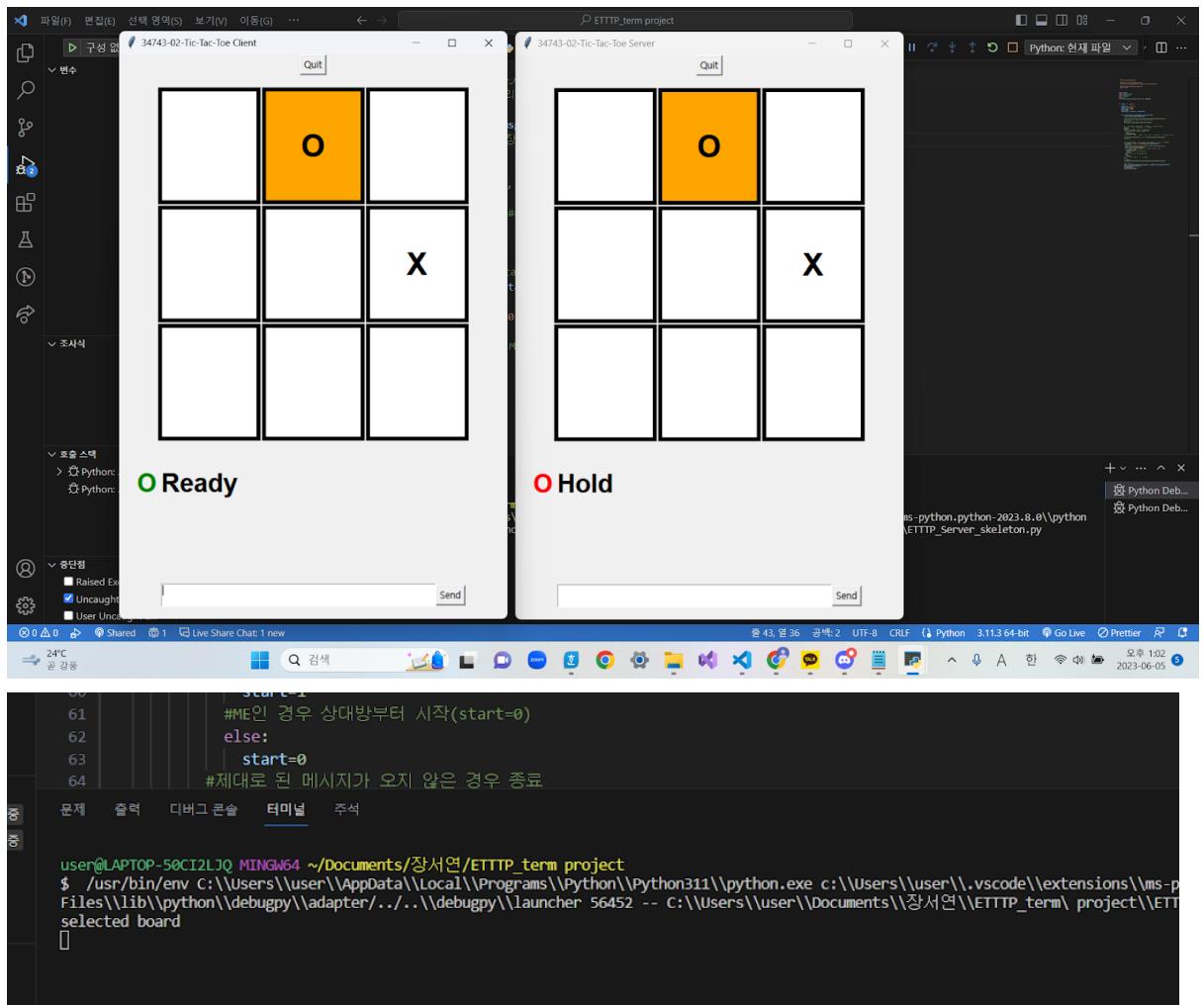


- 3) 자기 차례가 아닌데 메시지를 보낼 경우, 보드 업데이트가 되지 않고 콘솔창에 not my turn이 출력됩니다.



- 4) 이미 선택된 곳(2,1)을 다시 누를 경우, 보드에 변화가 없고 콘솔창에 selected board라고 출력되도록 하였습니다.

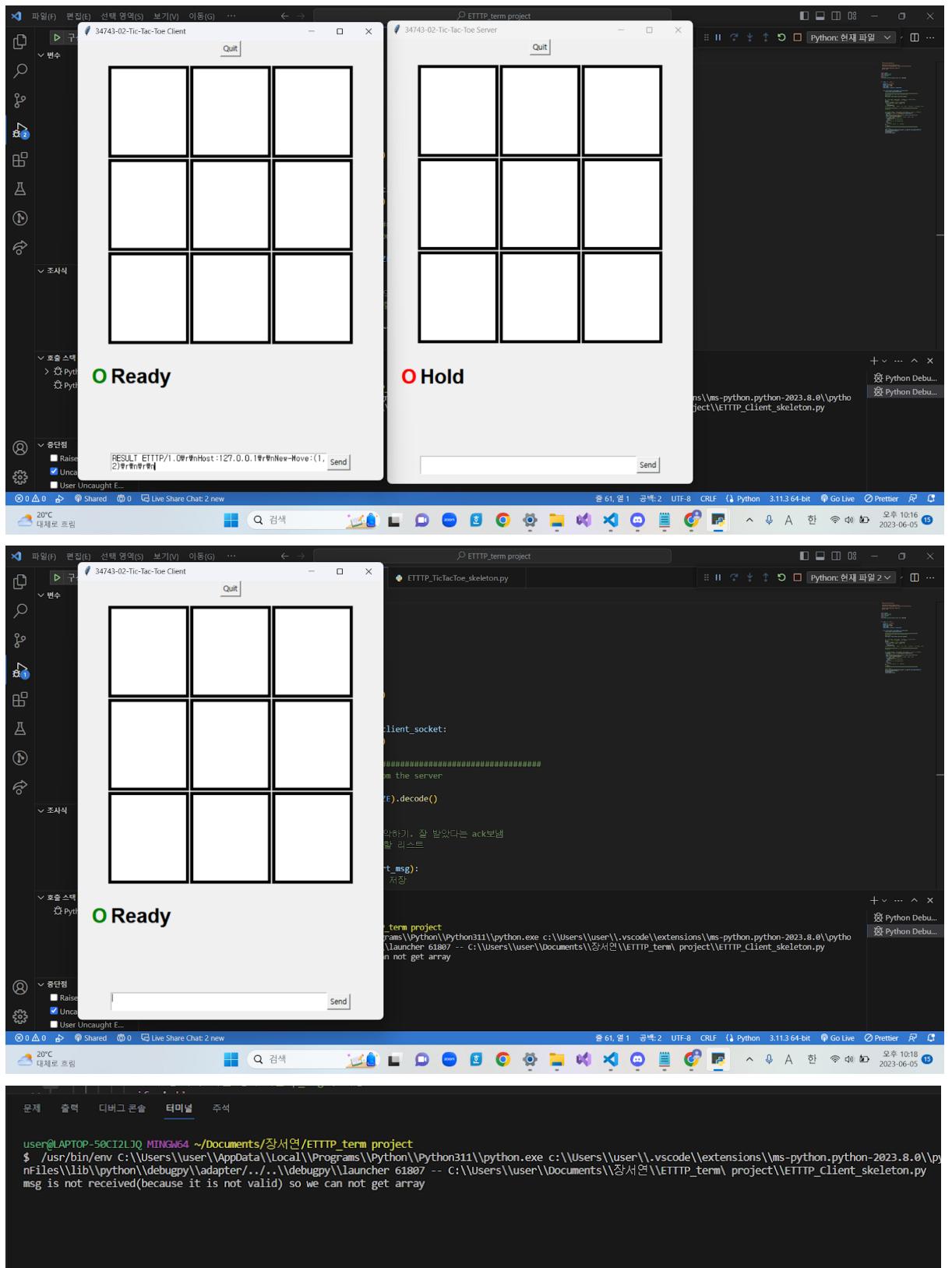




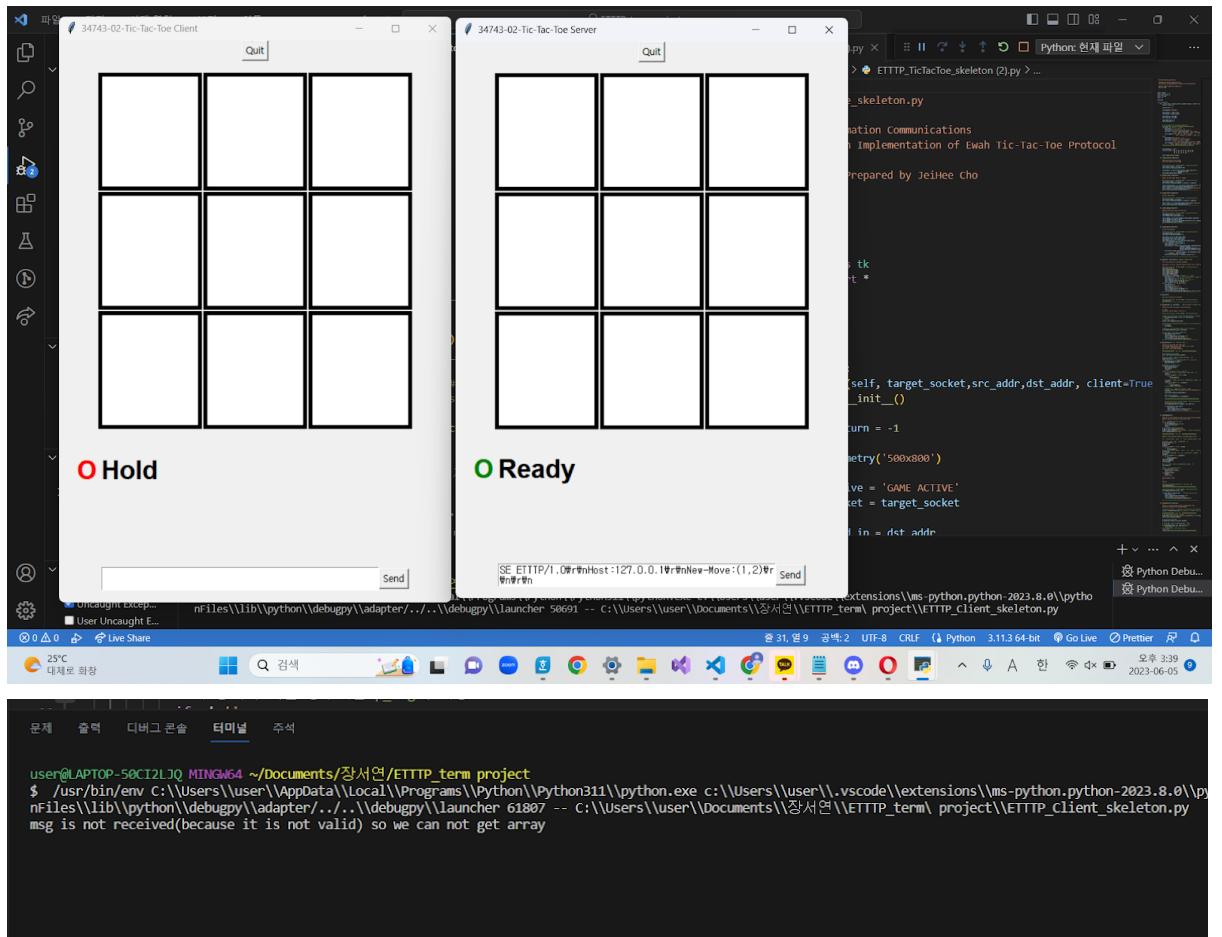
아래의 사진들은 ETTTP 형식이 아닌 다른 메세지를 입력했을 때의 화면들입니다. 저희는 여기에서 다른 메세지 형식(ex- RESULT, ETTP, 2.0, 128.13.0.1등)을 입력한 경우 msg가 전송되지 않도록 코드를 구현했기 때문에, msg를 받지 못한 peer는 check\_msg를 실행하게 되면 배열에 아무 입력도 들어가지 않아 out of range IndexError가 발생하게 되었고, 이를 해결하고자 check\_msg에 IndexError를 예외처리 했습니다. 그와 함께, 예외가 발생할 경우 터미널널에 msg is not received(because it is not valid) so we can not get array라는 문구가 출력되도록 구현했습니다.

- 5) SEND 대신에 RESULT를 입력하여 전송한 경우 상대 화면이 꺼지면서 게임이 종료되고, 터미널에는 msg is not received(because it is not valid) so we can not get array 가

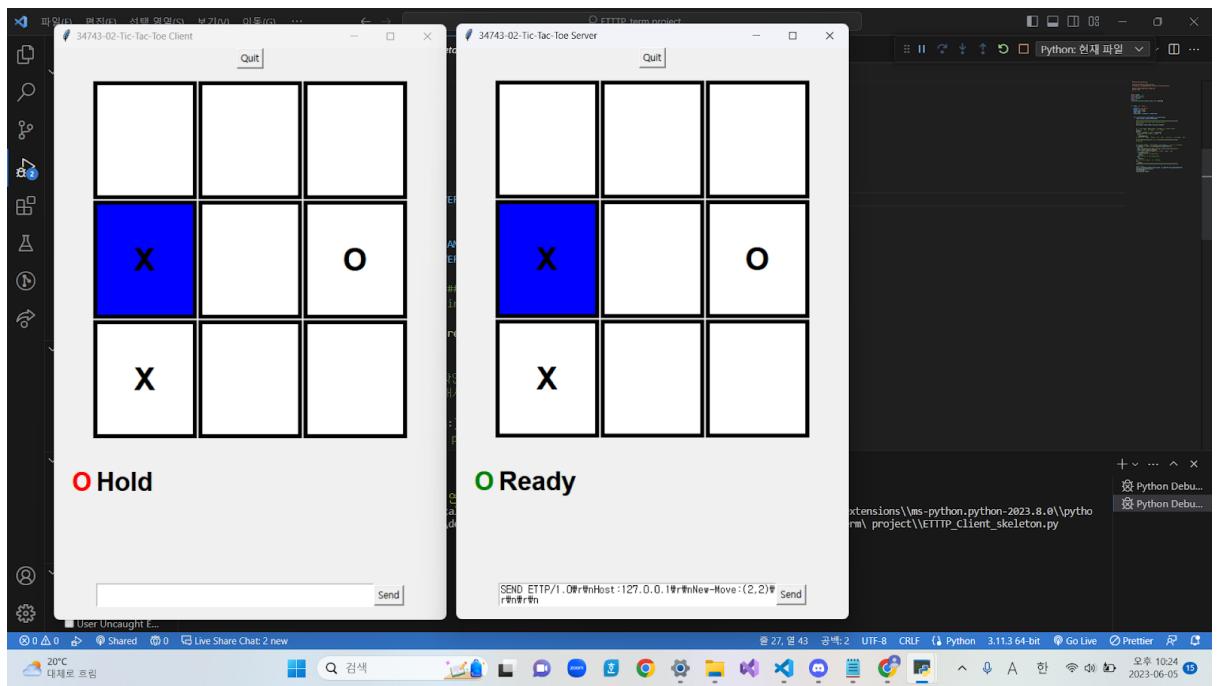
출력됩니다.



- 6) send 대신 se로 입력될 경우 또한 똑같이 게임이 종료되고 메시지가 터미널에 뜨도록 합니다.



- 7) ETTTP가 아닌 다른 메시지(ETTP)를 입력할 경우 상대방의 화면이 깨지면서 그대로 종료합니다. 또한 위와 같은 메세지가 터미널에 출력됩니다.



```

19
20
21 if __name__ == '__main__':
22
23 SERVER_IP = '127.0.0.1'
24 MY_IP = '127.0.0.1'
25 SERVER_PORT = 12000
26 SIZE = 1024
27 SERVER_ADDR = (SERVER_IP, SERVER_PORT)
28
29
30 with socket(AF_INET, SOCK_STREAM) as client_socket:
31     client_socket.connect(SERVER_ADDR)
32
33 ##### receive who will start first from the server #####
34 # Receive who will start first from the server
35 #start 보냄
36 start_msg = client_socket.recv(SIZE).decode()
37
38 #받은 msg 파싱해서 누가 시작인지 파악하기. 잘 받았다는 ack보냄
39 #p_msg: 받은 메시지를 파싱해서 저장할 리스트
40 p_msg=[]
41 for s in re.split('[\r\n:]',start_msg):
42     #s가 공백이 아닐 경우에만 p_msg에 저장

```

O Ready

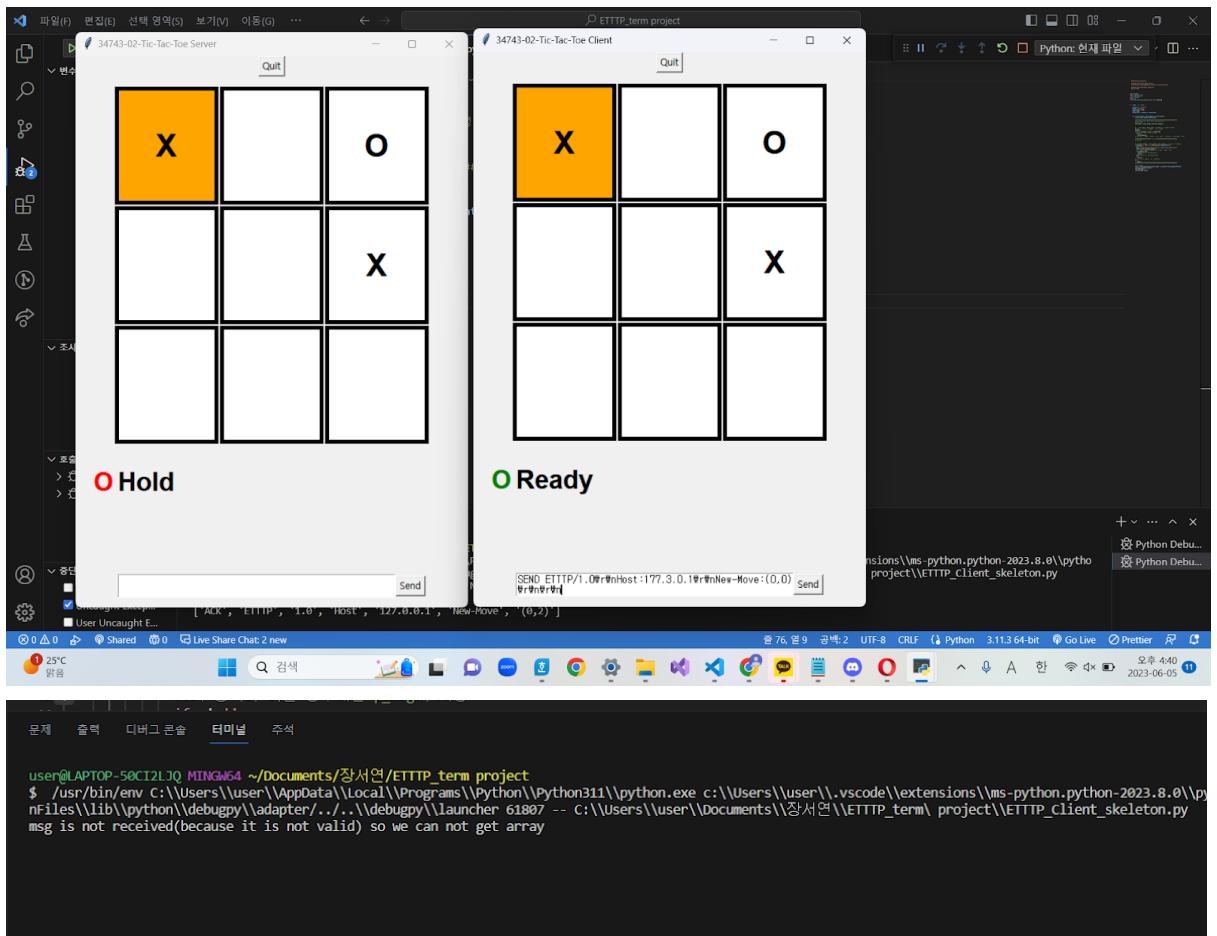
```

19
20
21 if __name__ == '__main__':
22
23 SERVER_IP = '127.0.0.1'
24 MY_IP = '127.0.0.1'
25 SERVER_PORT = 12000
26 SIZE = 1024
27 SERVER_ADDR = (SERVER_IP, SERVER_PORT)
28
29
30 with socket(AF_INET, SOCK_STREAM) as client_socket:
31     client_socket.connect(SERVER_ADDR)
32
33 ##### receive who will start first from the server #####
34 # Receive who will start first from the server
35 #start 보냄
36 start_msg = client_socket.recv(SIZE).decode()
37
38 #받은 msg 파싱해서 누가 시작인지 파악하기. 잘 받았다는 ack보냄
39 #p_msg: 받은 메시지를 파싱해서 저장할 리스트
40 p_msg=[]
41 for s in re.split('[\r\n:]',start_msg):
42     #s가 공백이 아닐 경우에만 p_msg에 저장

```

새 친구 풀기 (Ctrl+클릭) | ~/Documents/장서연/ETTTP\_term project  
\$ /usr/bin/env :/Users/user/AppData/Local/Programs/Python/Python311/python.exe c:/Users/user/.vscode/extensions/ms-python.python-2023.8.0/pythonFiles/lib/python/debugpy/adapter/.../debugpy/launcher 61899 -- C:/Users/user/Documents/장서연/ETTTP\_term/project/ETTTP\_Server\_skeleton.py  
msg is not received(because it is not valid) so we can not get array

8) ip주소가 다른 경우에도 전송이 되지 않는 것을 확인할 수 있었습니다.



1)~2)까지의 케이스를 통해 소켓 통신이 잘 됨을 확인할 수 있었고, 3)~8)의 케이스를 통해 디버그 메시지가 제대로 전송되지 않는 경우에도 게임이 종료되고 콘솔창에 출력되며 정상적으로 처리됨을 확인할 수 있었습니다.