

## Table of Contents

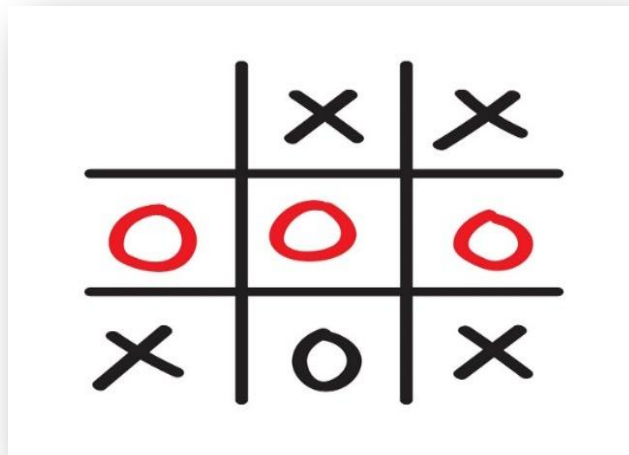
<b>1.0 Introduction .....</b>	<b>1</b>
<b>1.1 Key Features &amp; Functionalities .....</b>	<b>1</b>
<b>1.2 Advantages of using the socket programming.....</b>	<b>2</b>
<b>2.0 Import method emphasis.....</b>	<b>3</b>
<b>2.1 server.py implementation .....</b>	<b>3</b>
<b>2.2 client.py implementation.....</b>	<b>4</b>
<b>2.3 grid.py implementation .....</b>	<b>6</b>
<b>2.4 External Libraries.....</b>	<b>7</b>
2.4.1 Installing PyGame.....	7
2.4.2 Installing Tkinter .....	7
<b>2.5 Running the Files .....</b>	<b>8</b>
<b>2.6 How to Play .....</b>	<b>11</b>
<b>3.0 System interface Design.....</b>	<b>12</b>
<b>3.1 System Interface .....</b>	<b>12</b>
<b>3.2 Structure &amp; client-server communication.....</b>	<b>13</b>
<b>3.3 Implementation Issues .....</b>	<b>14</b>
<b>4.0 Conclusion.....</b>	<b>15</b>
<b>5.0 Appendix.....</b>	<b>16</b>

## 1.0 Introduction

**Cross and Not** (also known as Tic Tac Toe) is a game in which two players take turns drawing either an 'X' or an 'O' in one of nine cells on a grid. The player that obtains three of the identical symbols in a row either horizontally, vertically or diagonally is declared the winner. It is a solved game, with a forced draw assuming best play from both players.

In this scenario, the group has decided to implement the Cross and Not Game using raw TCP/IP sockets between the client and server. There are two roles involved in this game, Client and server. The client acts like a normal user to the game, which request the server to start the game and plays his/her represented symbols on the grid. The server will do the following tasks : accept client's request for connection , response to client's request , check winning or draw condition during the gaming process , display the results , and much more.

The following diagram showed the sample strcture of a Tic Tac Tac game.



### 1.1 Key Features & Functionalities

1. The program demonstrates Client-Server communication using sockets programming.
2. It consisted the structure of 3x3 grid with 9 cells which allows the client and server to places their represented symbols and compete.
3. It has user friendly application Interface with point and click to mark player moves
4. It is a single player game, which clients compete with the server. However, the user is also representing the server to marks the moves. This helps the computer beginner in better understand the client-server communication.
5. Alert dialogue box will appear for when there is a winner or if there is a draw

6. Invalid move check whereby a player can only put a symbol in a cell that does not contain another symbol.
7. The program consists of Game Reset and Escape window feature which allows the user to terminate the game.

### **1.2 Advantages of using the socket programming**

- Using socket programming while building the cross and not application meant that each component such as the client, server and the actual game had to be implemented separately which allowed us to focus on performing one individual task at a time which makes the game easier to develop and maintain.
- Sockets allow for having a central place where all communications flow which makes it easier to manage the communication and access control.
- Using TCP sockets ensures the data sent will not be lost or corrupt while sending and receiving data.

## 2.0 Import method emphasis

### 2.1 server.py implementation

```
SERVER = socket.gethostname(socket.gethostname())
PORT = 5050
connection_established = False
conn, addr = None, None

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind((SERVER, PORT))
sock.listen(1)
```

These lines of code are for creating a TCP socket for the server whereby the SERVER is set to get the local host of the user and PORT:5050 is the port that the server is listening on. The arguments passed to socket() are AF\_INET which is used for IPV4 and SOCK\_STREAM is the socket type for TCP protocol. The bind() function is used to associate the socket with the localhost and a port number.

```
def create_thread(target):
    thread = threading.Thread(target=target)
    thread.daemon = True
    thread.start()
```

A separate thread is created to send and receive data from the client so without putting the data into a separate thread it will block the main thread and the game window will not respond.

```
def receive_data():
    global turn
    while True:
        # receive data from the client, it is a blocking method
        data = conn.recv(1024).decode()
        # the format of the data after splitting is: ['x', 'y', 'yourturn', 'playing']
        data = data.split('-')
        x, y = int(data[0]), int(data[1])
        if data[2] == 'yourturn':
            turn = True
        if data[3] == 'False':
            grid.game_over = True
        if grid.get_cell_value(x, y) == 0:
            grid.set_cell_value(x, y, '0')
```

The receive\_data() function receives data from the client.

```
def waiting_for_connection():
    global connection_established, conn, addr
    print('[STATUS]', f'server is listening on port {PORT}')
    print('[STATUS]', 'server is waiting for connection.....')
    conn, addr = sock.accept() # wait for a connection, it is a blocking method
    print('[STATUS]', 'client is connected')
    connection_established = True
    receive_data()
```

This function allows for the client to be connected to the server and data to be sent and received.

## 2.2 client.py implementation

```
SERVER = socket.gethostname(socket.gethostname())
PORT = 5050

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((SERVER, PORT))
```

These lines of code are for creating a TCP socket for the client whereby the SERVER is set to get the user's local host IP and connect it to the same port as the server, which it is listening from (PORT: 5050).

```
def create_thread(target):
    thread = threading.Thread(target=target)
    thread.daemon = True
    thread.start()
```

A threading module is used to prevent `recieve_data()` function from blocking the main thread so without putting the data into a separate thread it will block the main thread and the game window will not respond.

```
def receive_data():
    global turn
    while True:
        # receive data from the server, it is a blocking method
        data = sock.recv(1024).decode()
        # the format of the data after splitting is: ['x', 'y', 'yourturn', 'playing']
        data = data.split('-')
        x, y = int(data[0]), int(data[1])
        if data[2] == 'yourturn':
            turn = True
        if data[3] == 'False':
            grid.game_over = True
        if grid.get_cell_value(x, y) == 0:
            grid.set_cell_value(x, y, 'X')
```

This `recieve_data()` function receives data from the server.

```
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.MOUSEBUTTONDOWN and not grid.game_over: # left mouse click down triggers event 0
            if pygame.mouse.get_pressed()[0]:
```

In our game the client will always play as 'O' so therefore the above is for registering the left mouse button click to the grid - if `event.type == pygame.MOUSEBUTTONDOWN`.

## 2.3 grid.py implementation

```
class Grid:
    def __init__(self):
        self.grid_lines = [((0, 200), (600, 200)), # first horizontal line
                           ((0, 400), (600, 400)), # second horizontal line
                           ((200, 0), (200, 600)), # first vertical line
                           ((400, 0), (400, 600))] # second vertical line

    def draw(self, surface):
        for line in self.grid_lines:
            pygame.draw.line(surface, (200, 200, 200), line[0], line[1], 2)
```

These lines of code are used to draw the grid lines of the cross and not grid which is created using a list of tuples; (x,y) coordinates. Pygame.draw.line() is a pygame function to draw coordinates.

```
def is_within_bounds(self, x, y):
    return x >= 0 and x < 3 and y >= 0 and y < 3
```

This function checks whether the mouse click has been placed in a valid position i.e the cell is not already taken by another symbol.

```
def check_grid(self, x, y, player):
    count = 1
    for index, (dirx, diry) in enumerate(self.search_direction):
        if self.is_within_bounds(x+dirx, y+diry) and self.get_cell_value(x+dirx, y+diry) == player:
            count += 1
            xx = x + dirx
            yy = y + diry
            if self.is_within_bounds(xx+dirx, yy+diry) and self.get_cell_value(xx+dirx, yy+diry) == player:
```

This function checks if the mouse click has been placed within the bounds of the grid in a valid position and begins to use the search algorithm(search\_direction) to check the surrounding cells for the same symbol. The search algorithm checks in directions; North, North west, West, South West, South East, East and North East. If the same symbol has been found in one of the directions then the algorithm will check the opposite direction of the symbol to check whether the same symbol is on the opposite side as well.

```
if count == 3: # Win condition 3 in a row
    messagebox.showinfo("Game Over", f"{player} Wins!")

    window.quit()
```

This is to check the winning condition. If three of the same symbols are in a row then either X or O wins, a message box will pop out and display the winning player.

## 2.4 External Libraries

We have used two external libraries, the first being pygame which was used to create the functionalities of the cross and not game such as the grid and events like mouse clicks. The second library used was tkinter which is used for creating pop up windows when a player wins or if there is a draw. Before installing pygame and tkinter, please ensure you have installed python 3.7 or higher and pip is installed

### 2.4.1 Installing PyGame

1. First check whether pip is installed by opening Command Line and entering:

**pip --version**

If pip is not installed, install it by following the steps in this link:

<https://phoenixnap.com/kb/install-pip-windows>

2. To install pygame enter the command:

**pip install pygame**

3. Verify pygame installation by entering:

**>python**

into the command line and then entering:

**>> import pygame**

### 2.4.2 Installing Tkinter

1. Ensure pip is installed
2. Open command line and enter:

**pip install tk**



## 2.5 Running the Files

Once all external libraries are installed, open the assignment folder [folder\_name] in your IDE of choice. We will be using Visual Studio Code with Python 3.9.2

Figure 2.5.1: Run the file named server.py

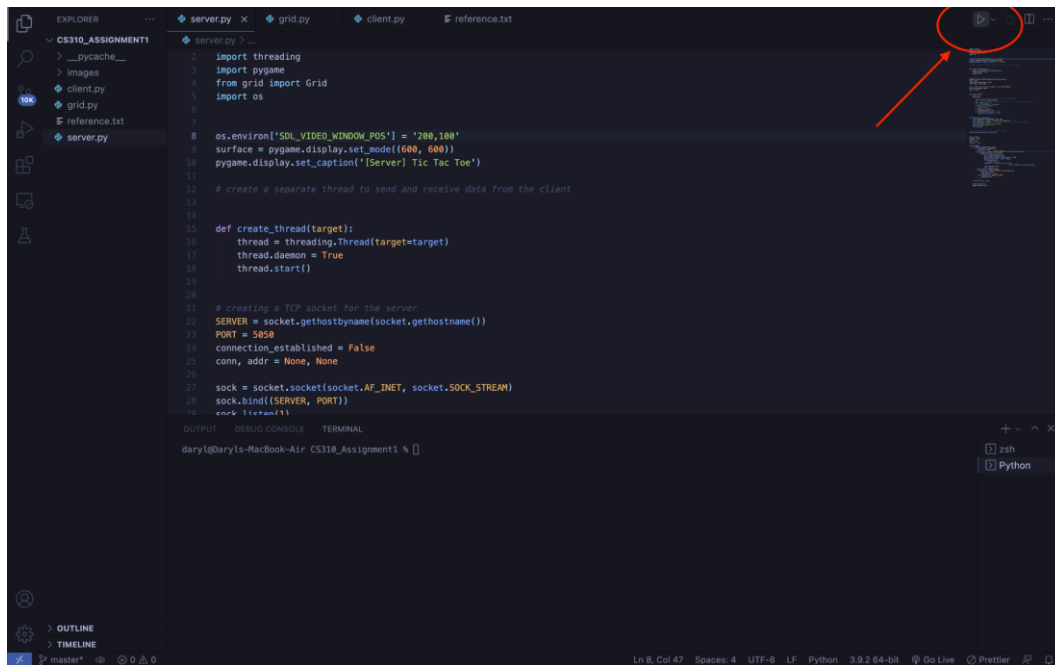


Figure 2.5.2: Server GUI Opens upon running server.py

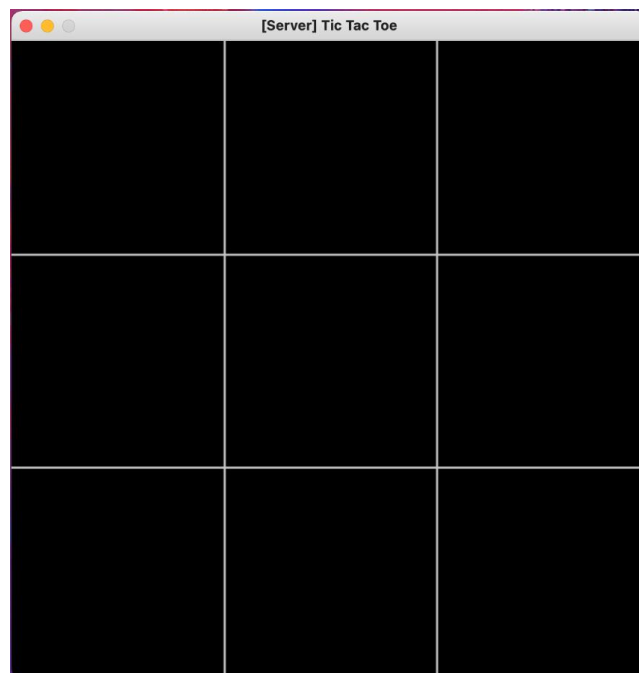
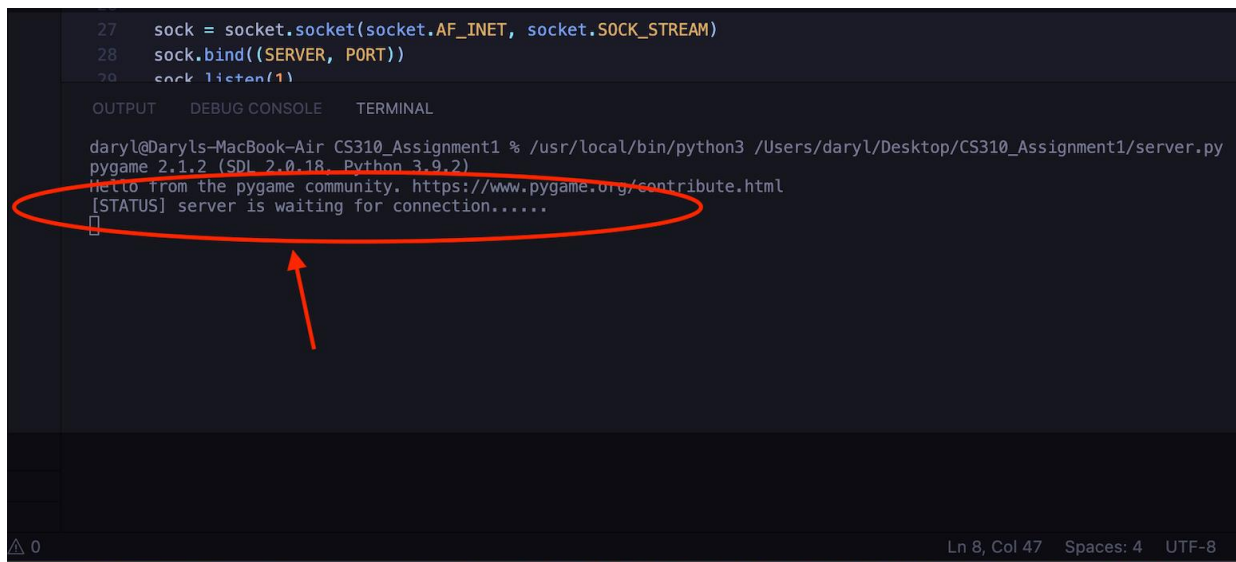


Figure 2.5.3: Terminal will show that the server is waiting for Connection

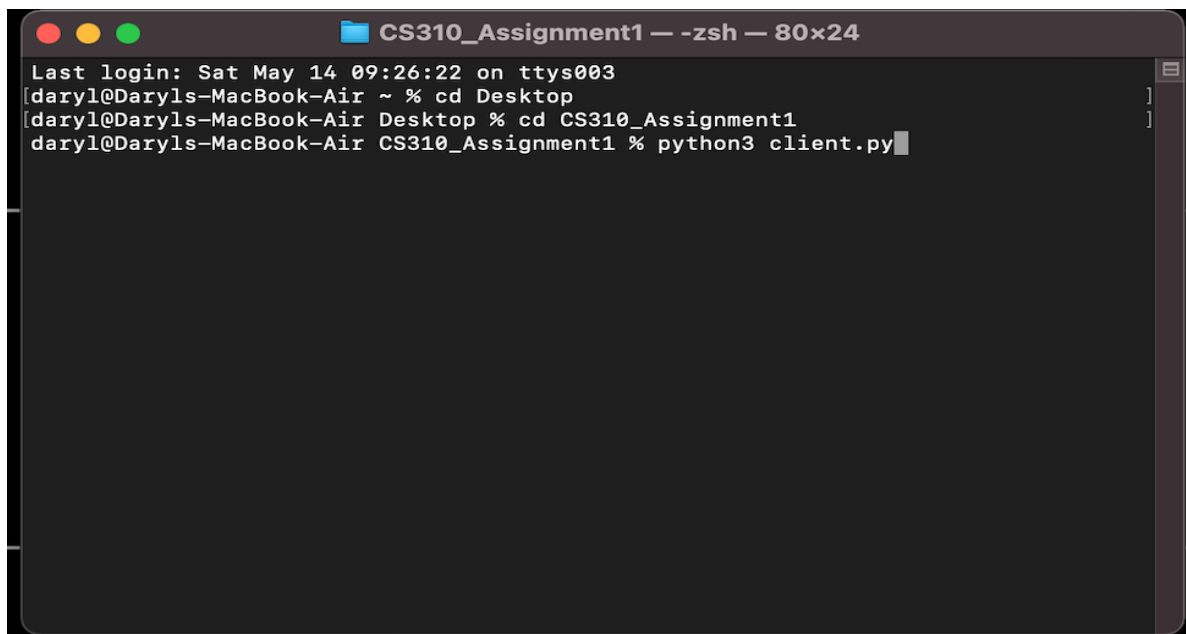
A terminal window with a dark background. At the top, there are three tabs: 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active. The terminal shows the command `/usr/local/bin/python3 /Users/daryl/Desktop/CS310_Assignment1/server.py` being executed. The output is: `pygame 2.1.2 (SDL 2.0.18, Python 3.9.2)`, `Hetto from the pygame community. https://www.pygame.org/contribute.html`, and `[STATUS] server is waiting for connection.....`. The last line is circled in red, and a red arrow points to it from below. The bottom status bar shows `Ln 8, Col 47 Spaces: 4 UTF-8`.

```
27 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
28 sock.bind((SERVER, PORT))
29 sock.listen(1)

OUTPUT  DEBUG CONSOLE  TERMINAL

daryl@Daryls-MacBook-Air CS310_Assignment1 % /usr/local/bin/python3 /Users/daryl/Desktop/CS310_Assignment1/server.py
pygame 2.1.2 (SDL 2.0.18, Python 3.9.2)
Hetto from the pygame community. https://www.pygame.org/contribute.html
[STATUS] server is waiting for connection.....
█
```

Figure 2.5.4: Run client.py using the command line

A terminal window titled `CS310_Assignment1 — zsh — 80x24`. It shows a sequence of commands: `cd Desktop`, `cd CS310_Assignment1`, and `python3 client.py`. The cursor is at the end of the last command. The window has standard macOS window controls (red, yellow, green buttons) at the top left.

```
Last login: Sat May 14 09:26:22 on ttys003
daryl@Daryls-MacBook-Air ~ % cd Desktop
daryl@Daryls-MacBook-Air Desktop % cd CS310_Assignment1
daryl@Daryls-MacBook-Air CS310_Assignment1 % python3 client.py█
```

Figure 2.5.5: Client GUI Opens upon running client.py

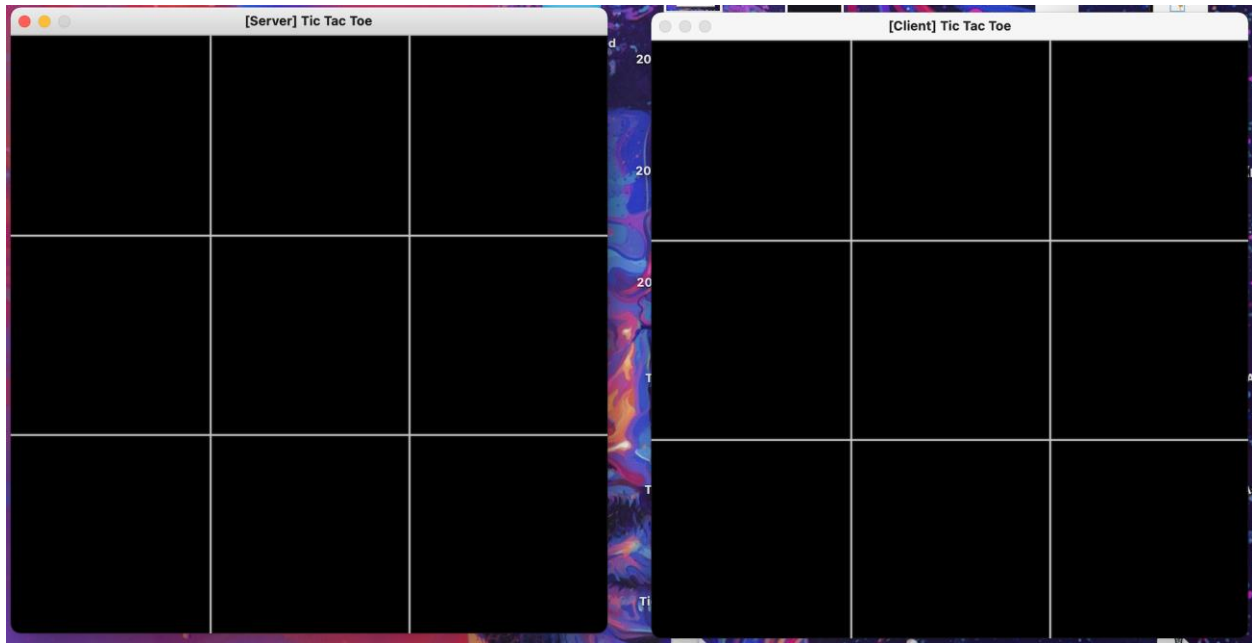


Figure 2.5.6: Terminal will show that the client is now connected to the server

```
28 sock.bind((SERVER, PORT))
29 sock.listen(1)

OUTPUT  DEBUG CONSOLE  TERMINAL

daryl@Daryls-MacBook-Air CS310 Assignment1 % /usr/local/bin/python3 /Users/daryl/Desktop/C
pygame 2.1.1.2 (SDL 2.0.18, Python 3.9.2)
Hello from the pygame community. https://www.pygame.org/contribute.html
[STATUS] server is waiting for connection.....
[STATUS] client is connected
```

The game is now ready to be played between the server [Player X] and the client [Player O]

**Note:** Server will always have the first turn.

## 2.6 How to Play

Figure 2.6.1: Player 1 [X] can point and click on any of the cells of the [Server] Tic-Tac-Toe Window to mark their turn. Player 1 [X] will be representing the server and make response to the client (Player 2 [O]). The Client window will immediately receive the data and display Player 1's move.

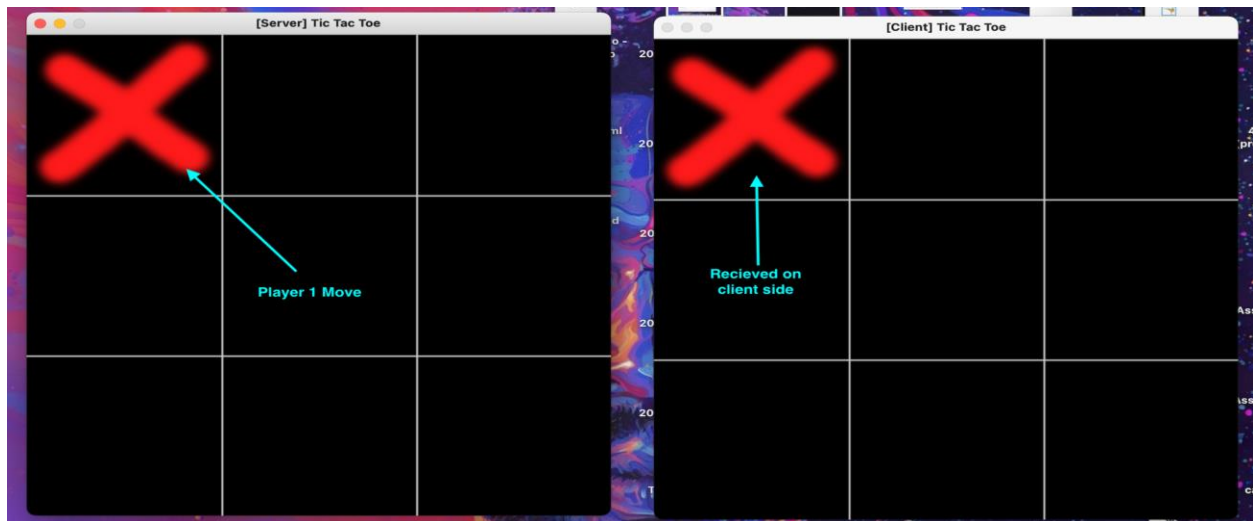


Figure 2.6.2: Player 2 [O] can point and click on any of the remaining cells of the [Client] Tic-Tac-Toe Window to mark their turn. The Server window will immediately receive the data and display Player 2's move.



Additional functionality: Press ESC key to close each window and Press SPACE bar to reset

## 3.0 System interface Design

### 3.1 System Interface

Figure 3.2.1: Server-Side Graphical User Interface

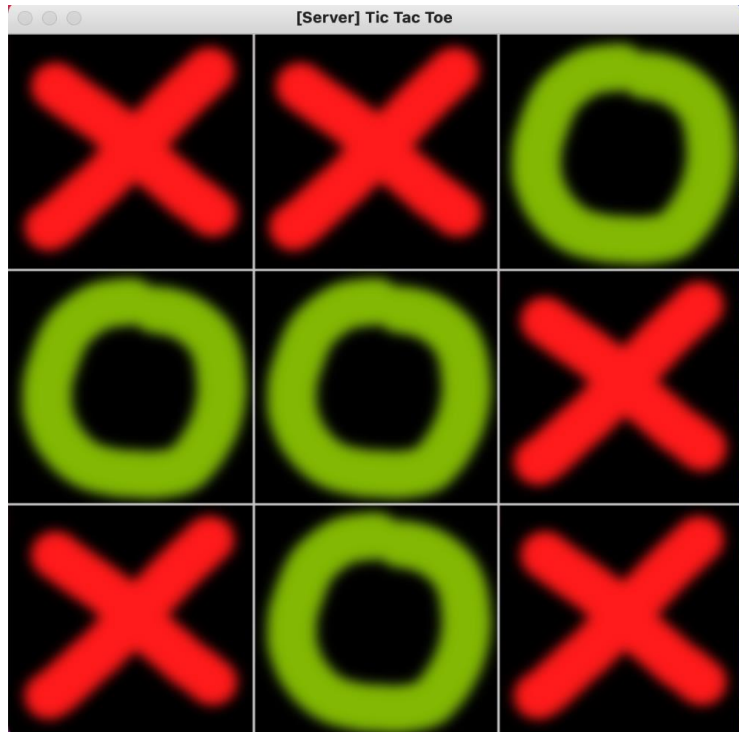
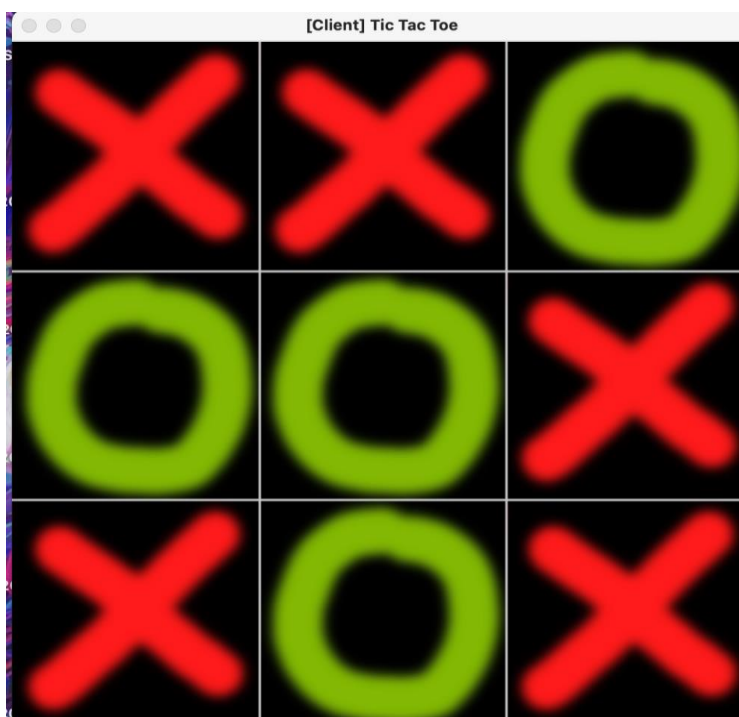
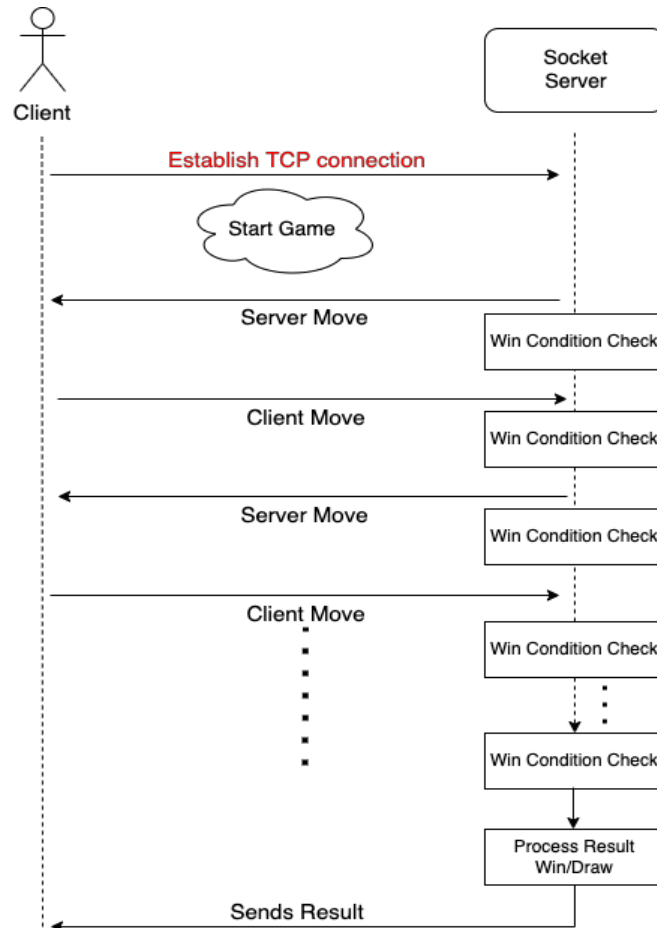


Figure 3.2.2: Client-Side Graphical User Interface



### 3.2 Structure & client-server communication

The structure of our Cross or Not Application are shown as follow:



The client-server communication will explain the above structure. The client will first initiate the TCP connection with Server. Once the server is connected, it is ready to receive the message. However, in our program, the server is the one which starts the game. The server will choose the grid and places [X] to starts the game. This grid which marked as [X] will simultaneously appears in client's interface. Then the server will conduct a Winning condition check, to check whether the game has winner. If no winner or draw condition match, then it will go to client's turn, the client will choose the grid and places [O]; This grid marked as [O] (can be seen as a client request), will goes back to the server, and appears in the server's interface. Once the server receives the client's request, it will process winning or draw condition check again. If none of the condition matched, the processes will be repeated.

Once the winning or drawing condition is matched, the server will process the result and sends the result back to the client as a pop-up window message.