Game Description

Short Summary: In this game, 3 players compete against each other to make a row of 5 pebbles first.

How the game works:
Each player is assigned a color and takes turns placing 1 colored pebble on the board. Pebbles can only be placed where two lines intersect and cannot be placed on other pebbles.

Players want to make a row of 5 using their own colored pebbles while stopping other players from achieving the same goal. The row of 5 may be horizontal, vertical, or diagonal.

The game ends when any player makes a row of 5.

Design Description

General Information

Type of data transportation used: TCP
Language used: C#
Compiler used: visual studio
Windows version: win 11

How actions/messages are sent

- The client's actions are sent in the form of a string starting with the type of data then the data follows. The type of data string is a token used by clients to differentiate different types of messages from the server. The string is sent to the server who then sends it to all clients.

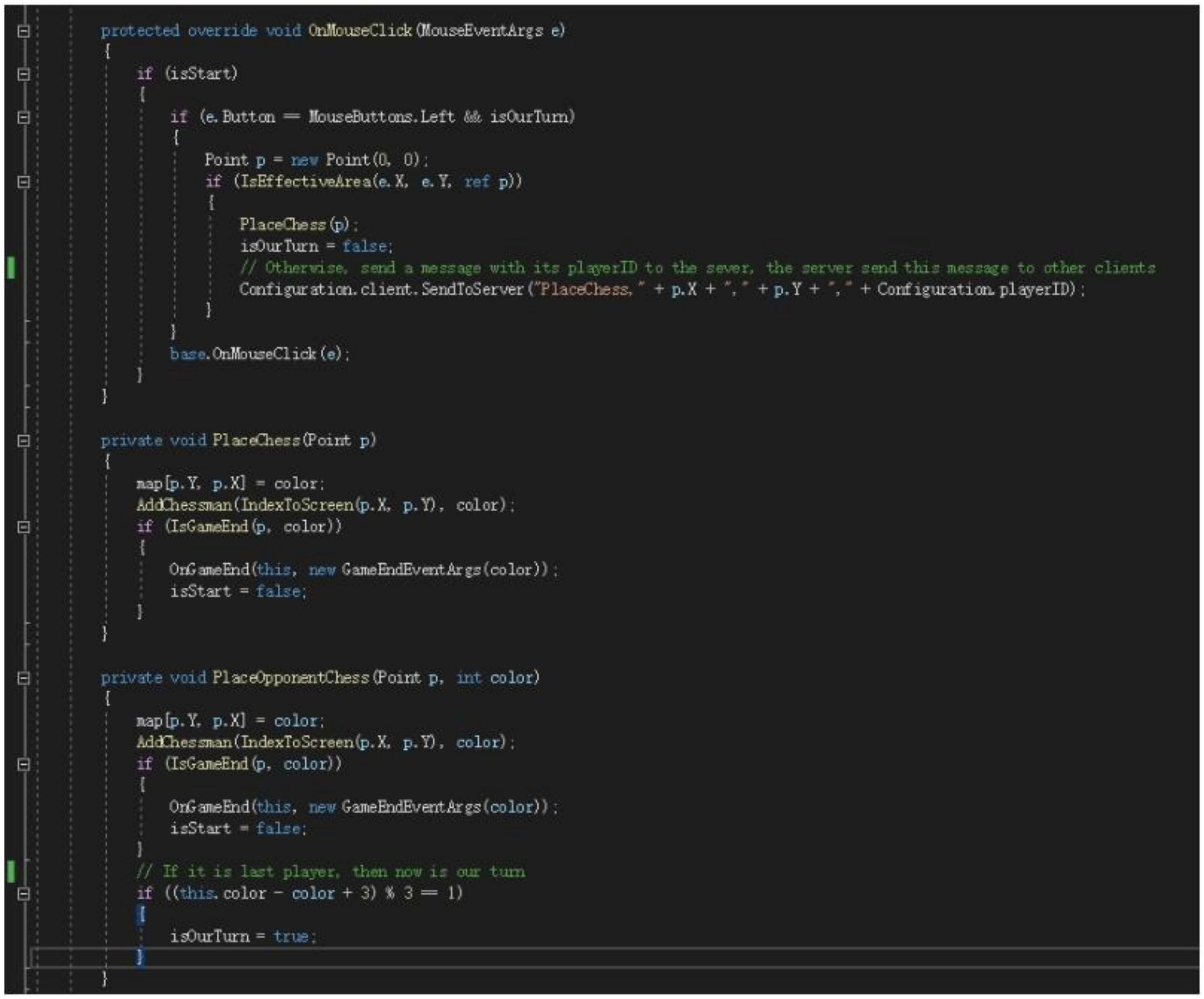
Example of a message string:

How messages are received

- Clients/Server split the message string to extract the token and place it through switch-case statements.

How the shared object is handled

- The shared objects are the crossings on the chessboard. When it is one player’s turn, the other players are not allowed to set any pebble on the chessboard i.e. it is locked, only the player in turn can handle the chessboard.



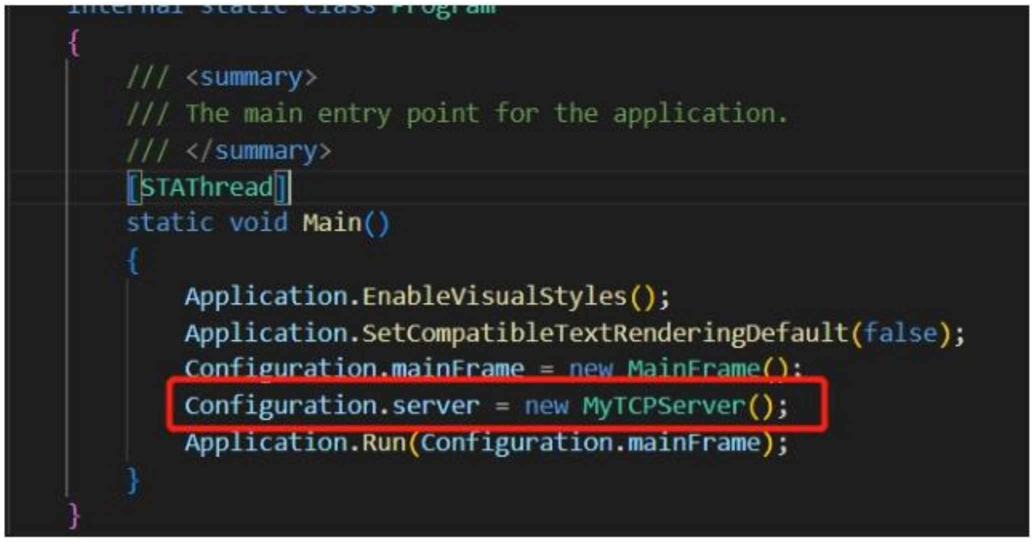
```
protected override void OnMouseClick(MouseEventArgs e)
{
    if (isStart)
    {
        if (e.Button == MouseButtons.Left && isOurTurn)
        {
            Point p = new Point(0, 0);
            if (IsEffectiveArea(e.X, e.Y, ref p))
            {
                PlaceChess(p);
                isOurTurn = false;
                // Otherwise, send a message with its playerID to the sever, the server send this message to other clients
                Configuration.client.SendToServer("PlaceChess," + p.X + "," + p.Y + "," + Configuration.playerID);
            }
        }
        base.OnMouseClick(e);
    }
}

private void PlaceChess(Point p)
{
    map[p.Y, p.X] = color;
    AddChessman(IndexToScreen(p.X, p.Y), color);
    if (IsGameEnd(p, color))
    {
        OnGameEnd(this, new GameEndEventArgs(color));
        isStart = false;
    }
}

private void PlaceOpponentChess(Point p, int color)
{
    map[p.Y, p.X] = color;
    AddChessman(IndexToScreen(p.X, p.Y), color);
    if (IsGameEnd(p, color))
    {
        OnGameEnd(this, new GameEndEventArgs(color));
        isStart = false;
    }
    // If it is last player, then now is our turn
    if ((this.color - color + 3) % 3 == 1)
    {
        isOurTurn = true;
    }
}
```

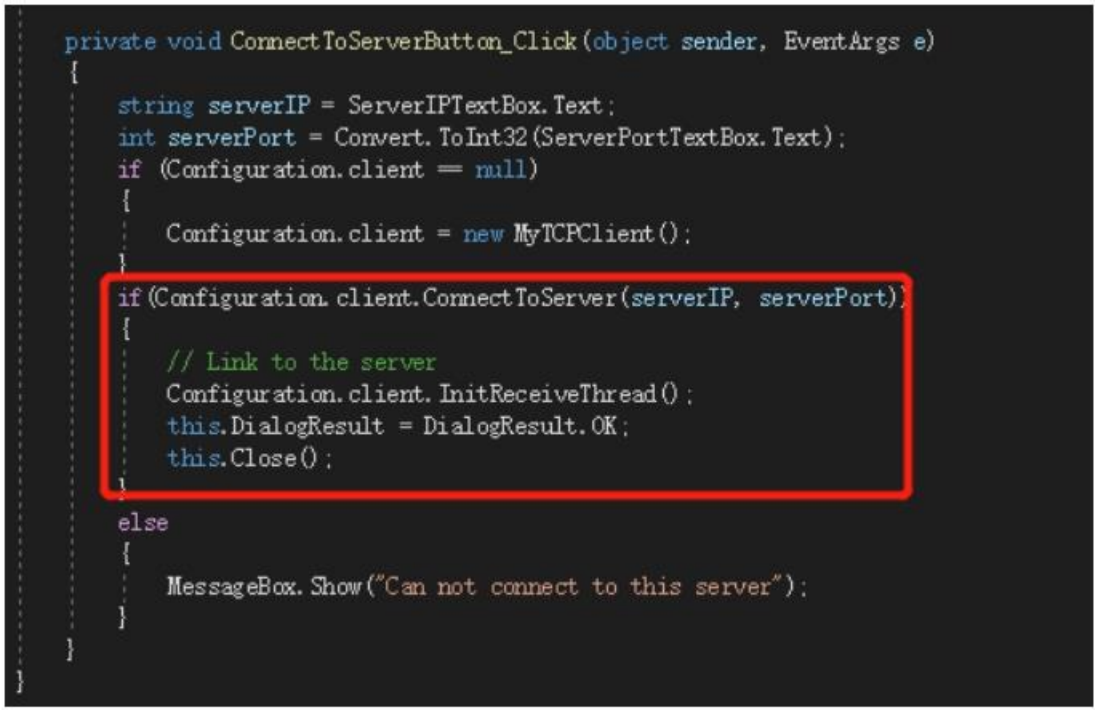
Socket Handling

- When the server starts to run, it also starts to listen to the socket connection request from client:



```
internal static class Program
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Configuration.mainFrame = new MainFrame();
        Configuration.server = new MyTCPServer();
        Application.Run(Configuration.mainFrame);
    }
}
```

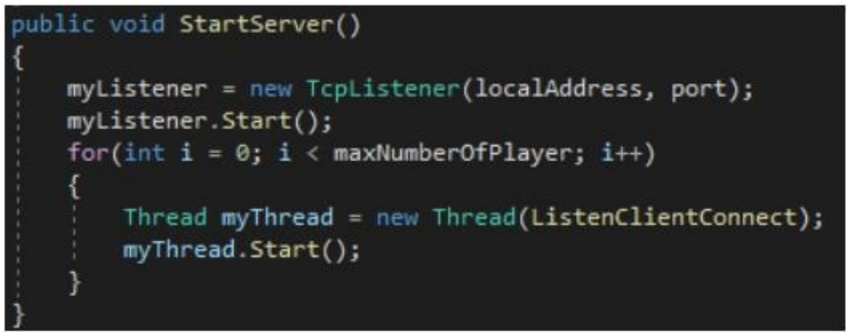
- While the client clicks to connect, it is trying to create a socket connection



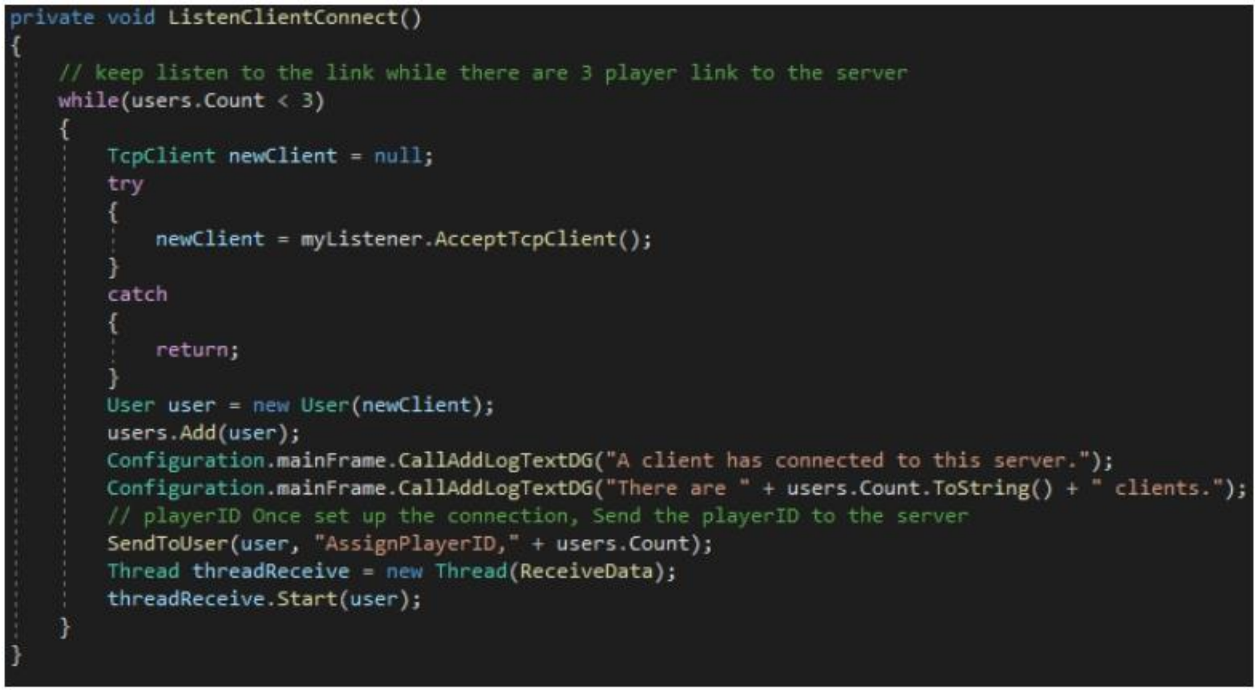
```
    private void ConnectToServerButton_Click(object sender, EventArgs e)
    {
        string serverIP = ServerIPTextBox.Text;
        int serverPort = Convert.ToInt32(ServerPortTextBox.Text);
        if (Configuration.client == null)
        {
            Configuration.client = new MyTCPClient();
        }
        if(Configuration.client.ConnectToServer(serverIP, serverPort))
        {
            // Link to the server
            Configuration.client.InitReceiveThread();
            this.DialogResult = DialogResult.OK;
            this.Close();
        }
        else
        {
            MessageBox.Show("Can not connect to this server");
        }
    }
}
```

Starting the game

- First the server needs to be launched. A TCP socket is set to the local IP address with port number 51888. Then threads will be created to listen for connecting clients. When the number of connected clients reaches 3, the server will stop listening to the connection request.

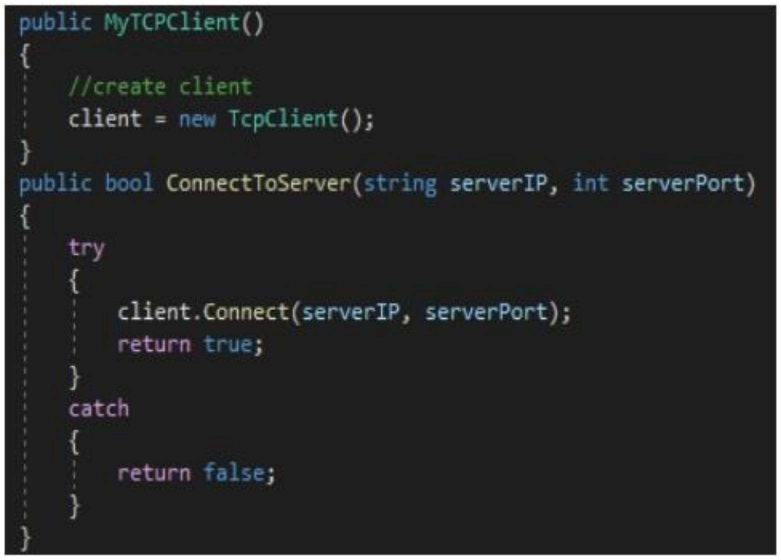


```
public void StartServer()
{
    myListener = new TcpListener(localAddress, port);
    myListener.Start();
    for(int i = 0; i < maxNumberOfPlayer; i++)
    {
        Thread myThread = new Thread(ListenClientConnect);
        myThread.Start();
    }
}
```

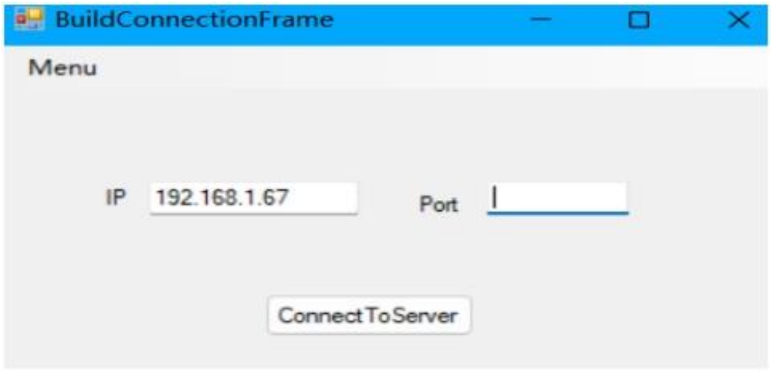


```
private void ListenClientConnect()
{
    // keep listen to the link while there are 3 player link to the server
    while(users.Count < 3)
    {
        TcpClient newClient = null;
        try
        {
            newClient = myListener.AcceptTcpClient();
        }
        catch
        {
            return;
        }
        User user = new User(newClient);
        users.Add(user);
        Configuration.mainFrame.CallAddLogTextDG("A client has connected to this server.");
        Configuration.mainFrame.CallAddLogTextDG("There are " + users.Count.ToString() + " clients.");
        // playerID Once set up the connection, Send the playerID to the server
        SendToUser(user, "AssignPlayerID," + users.Count);
        Thread threadReceive = new Thread(ReceiveData);
        threadReceive.Start(user);
    }
}
```

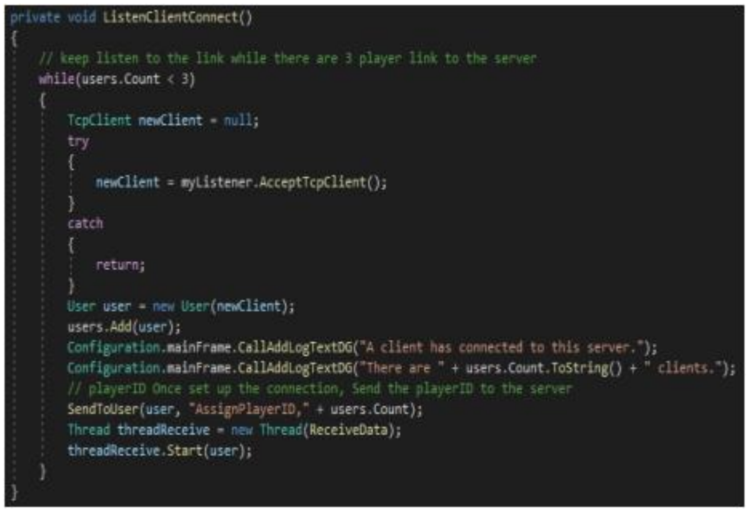
- Clients need to input the server IP and port number to connect.



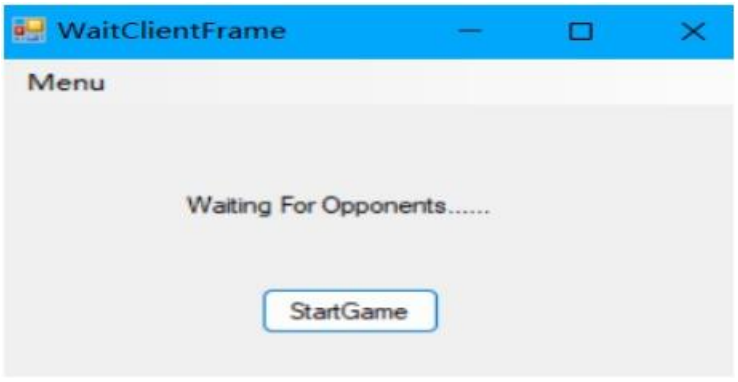
```
public MyTCPClient()
{
    //create client
    client = new TcpClient();
}
public bool ConnectToServer(string serverIP, int serverPort)
{
    try
    {
        client.Connect(serverIP, serverPort);
        return true;
    }
    catch
    {
        return false;
    }
}
```



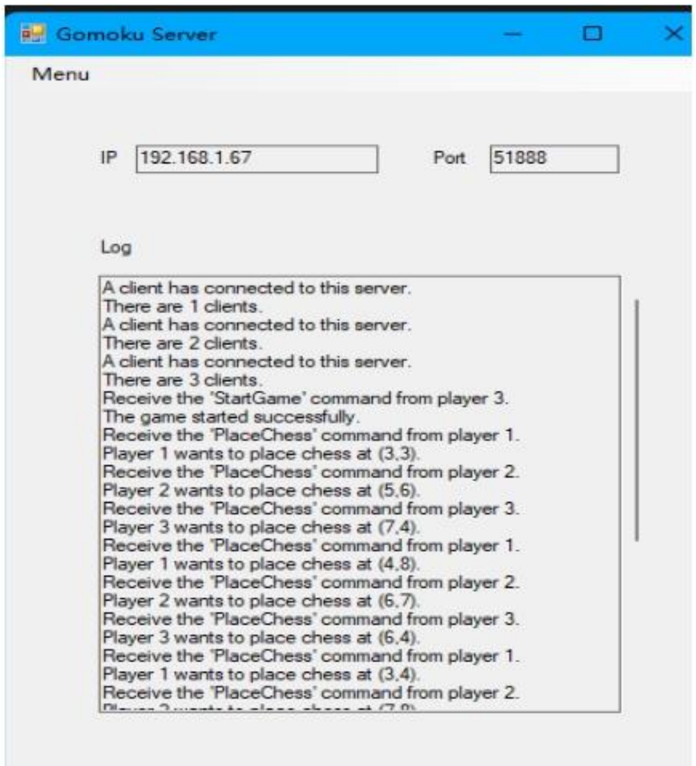
- Clients that successfully connect to the server are assigned (by the server) a player ID to differentiate the clients. The ID is sent to the respective client and another thread is made to listen for the client's actions.



```
private void ListenClientConnect()
{
    // keep listen to the link while there are 3 player link to the server
    while(users.Count < 3)
    {
        TcpClient newClient = null;
        try
        {
            newClient = myListener.AcceptTcpClient();
        }
        catch
        {
            return;
        }
        User user = new User(newClient);
        users.Add(user);
        Configuration.mainFrame.CallAddLogTextDG("A client has connected to this server.");
        Configuration.mainFrame.CallAddLogTextDG("There are " + users.Count.ToString() + " clients.");
        // playerID Once set up the connection, Send the playerID to the server
        SendToUser(user, "AssignPlayerID," + users.Count);
        Thread threadReceive = new Thread(ReceiveData);
        threadReceive.Start(user);
    }
}
```



- The server’s log will record the information such as connection application and player activity

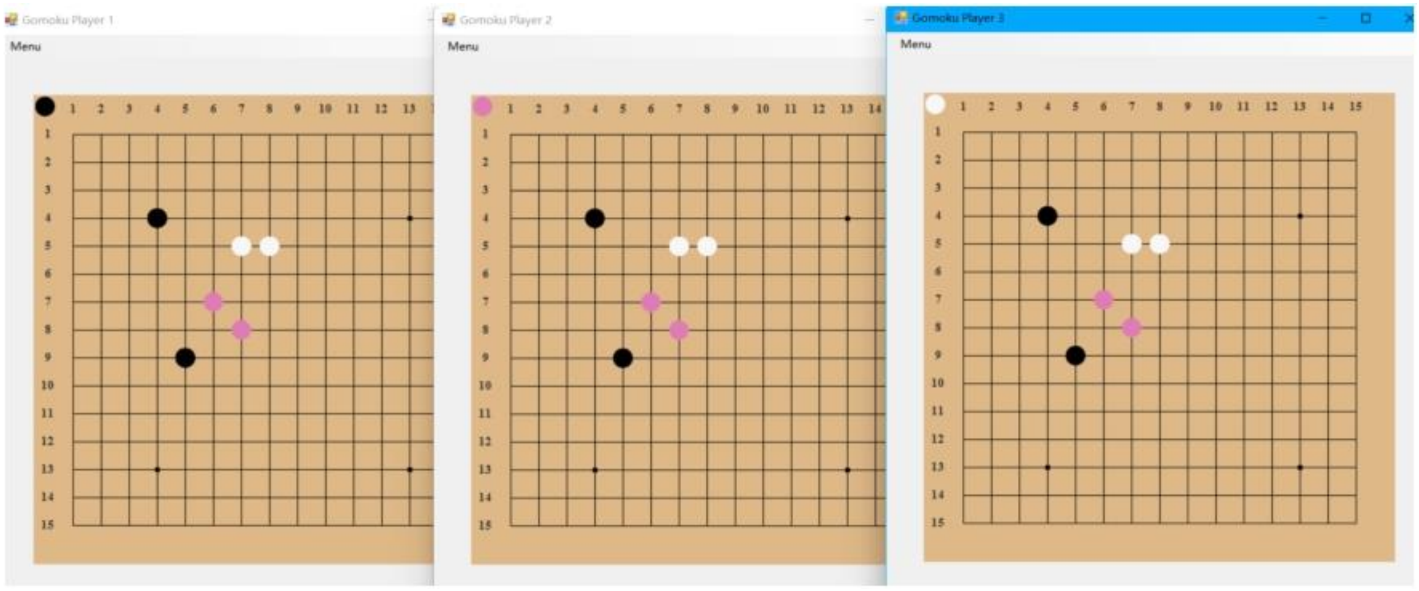


IP 192.168.1.67 Port 51888

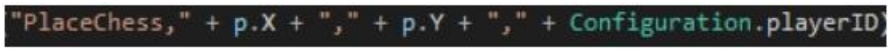
Log

```
A client has connected to this server.
There are 1 clients.
A client has connected to this server.
There are 2 clients.
A client has connected to this server.
There are 3 clients.
Receive the 'StartGame' command from player 3.
The game started successfully.
Receive the 'PlaceChess' command from player 1.
Player 1 wants to place chess at (3,3).
Receive the 'PlaceChess' command from player 2.
Player 2 wants to place chess at (5,6).
Receive the 'PlaceChess' command from player 3.
Player 3 wants to place chess at (7,4).
Receive the 'PlaceChess' command from player 1.
Player 1 wants to place chess at (4,8).
Receive the 'PlaceChess' command from player 2.
Player 2 wants to place chess at (6,7).
Receive the 'PlaceChess' command from player 3.
Player 3 wants to place chess at (6,4).
Receive the 'PlaceChess' command from player 1.
Player 1 wants to place chess at (3,4).
Receive the 'PlaceChess' command from player 2.
Player 2 wants to place chess at (7,8).
```

Playing the game

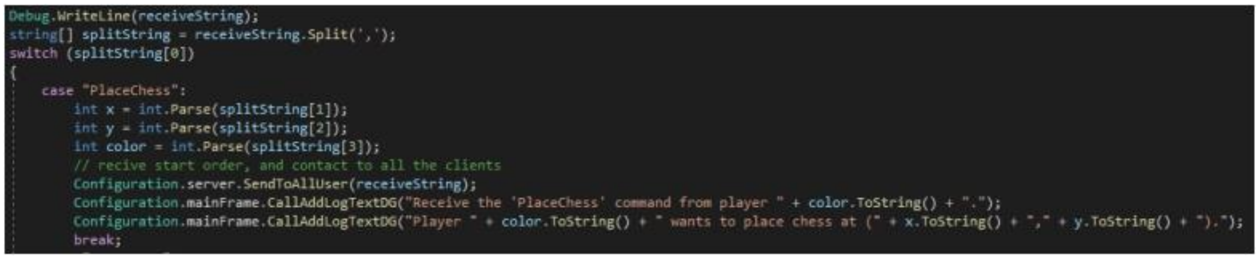


- The place a pebble in-game action is sent as a string in the order of:
  - type of action, x-coordinate, y-coordinate, and player ID.



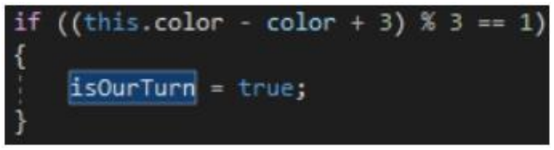
```
"PlaceChess," + p.X + "," + p.Y + "," + Configuration.playerID)
```

- When the client places a pebble, it is first executed on their application, then the same action is sent (in the form of a string as described above) to the server, who then sends it to all clients.



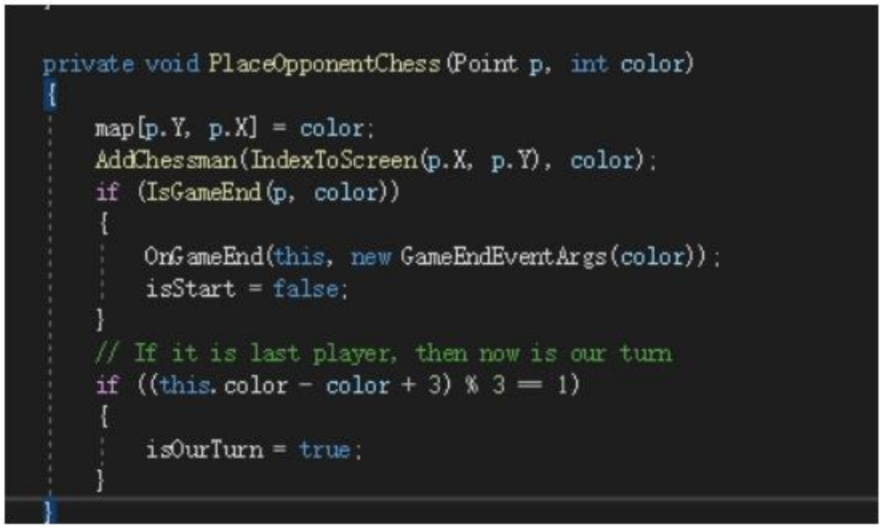
```
Debug.WriteLine(receiveString);
string[] splitString = receiveString.Split(',');
switch (splitString[0])
{
    case "PlaceChess":
        int x = int.Parse(splitString[1]);
        int y = int.Parse(splitString[2]);
        int color = int.Parse(splitString[3]);
        // recive start order, and contact to all the clients
        Configuration.server.SendToAllUser(receiveString);
        Configuration.mainFrame.CallAddLogTextDG("Receive the 'PlaceChess' command from player " + color.ToString() + ".");
        Configuration.mainFrame.CallAddLogTextDG("Player " + color.ToString() + " wants to place chess at (" + x.ToString() + "," + y.ToString() + ").");
        break;
```

- Only one player can place a pebble at a time. It is calculated using the player ID every time one of the other player's pebbles is placed.



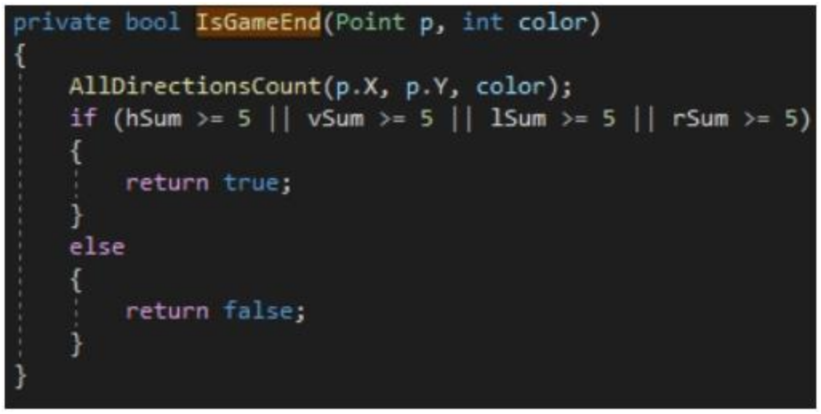
```
if ((this.color - color + 3) % 3 == 1)
{
    isOurTurn = true;
}
```

- Because the ID ranges from 1-3 if the last person to place a pebble is the player before the current client, then the current client knows it's their turn. The player that connected first goes first.



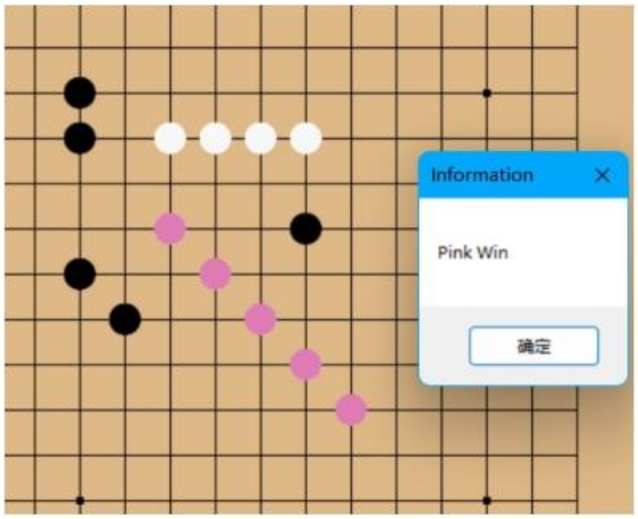
```
private void PlaceOpponentChess(Point p, int color)
{
    map[p.Y, p.X] = color;
    AddChessman(IndexToScreen(p.X, p.Y), color);
    if (IsGameEnd(p, color))
    {
        OnGameEnd(this, new GameEndEventArgs(color));
        isStart = false;
    }
    // If it is last player, then now is our turn
    if ((this.color - color + 3) % 3 == 1)
    {
        isOurTurn = true;
    }
}
```

- Everytime a pebble is placed, the game calculates whether a row of 5 is created. Thus, all clients will self calculate whether the game has ended and call the appropriate code to signal the game's end.



```
private bool IsGameEnd(Point p, int color)
{
    AllDirectionsCount(p.X, p.Y, color);
    if (hSum >= 5 || vSum >= 5 || lSum >= 5 || rSum >= 5)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

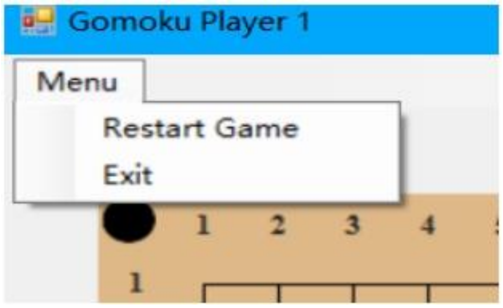
Ending the game

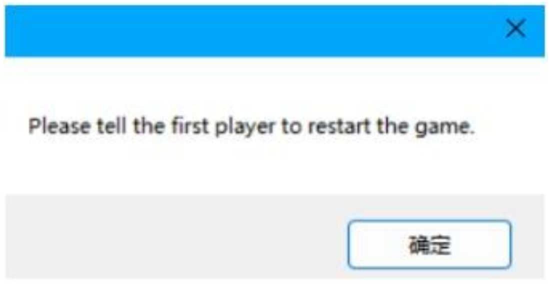

Information

Pink Win

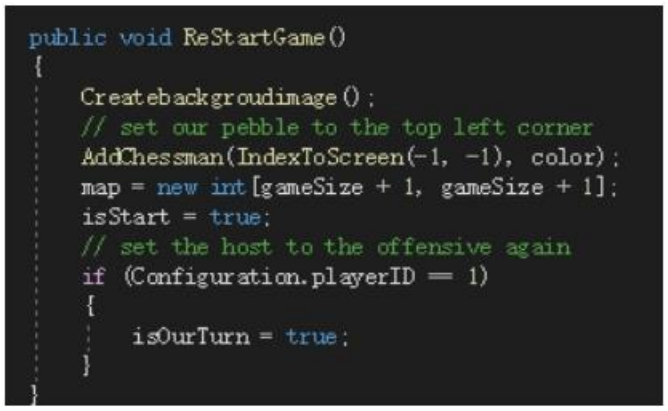
- If someone exits during the game, the other 2 clients will not be able to continue the game.

- If the game ends because someone wins, a pop up will appear announcing who won and the game may be restarted by player1 or exited.

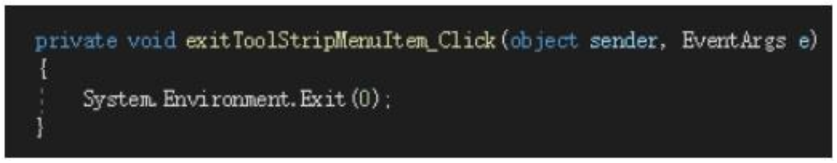



Please tell the first player to restart the game.

- Restart


```
public void ReStartGame()
{
    Createbackgroudimage();
    // set our pebble to the top left corner
    AddChessman(IndexToScreen(-1, -1), color);
    map = new int[gameSize + 1, gameSize + 1];
    isStart = true;
    // set the host to the offensive again
    if (Configuration.playerID == 1)
    {
        isOurTurn = true;
    }
}
```

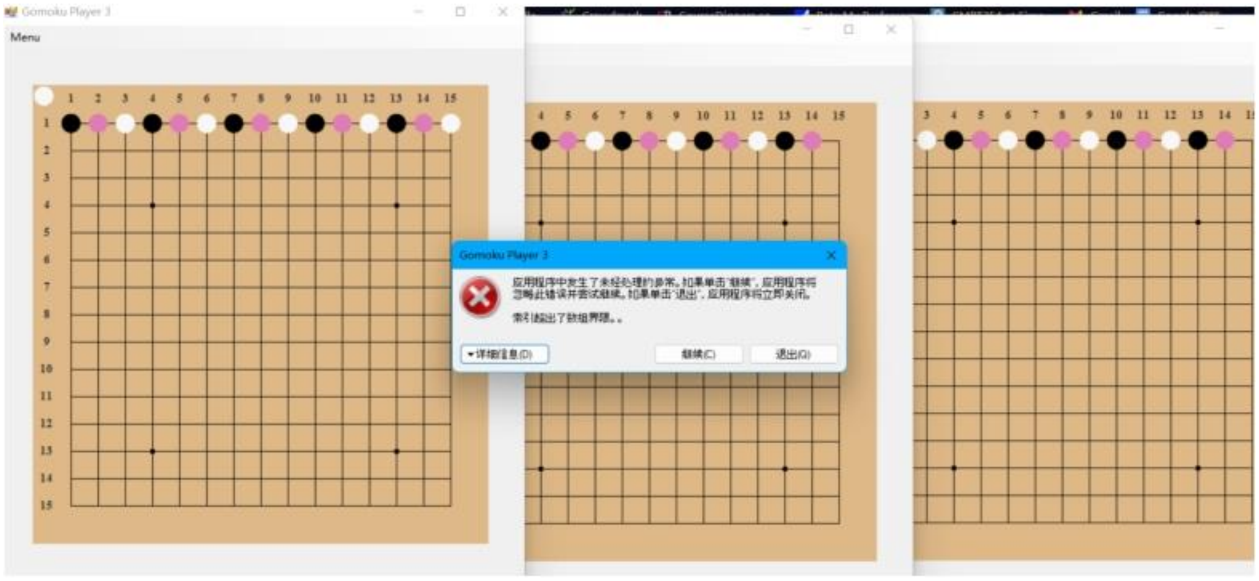
- Exit



```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    System.Environment.Exit(0);
}
```

Future Improvement

- Error when a line is fully set by pebbles



应用程序中发生了未经处理的异常。如果单击"继续"，应用程序将忽略此错误并尝试继续。如果单击"退出"，应用程序将立即关闭。

索引超出了数组界限。。

- Sometimes the player who starts the game exits mistakenly just after the game started
- Sometimes when someone exits the game in half, the rest players are able