BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

Giảng viên hướng dẫn: **ThS. VÕ TIẾN AN**

Sinh viên thực hiện:

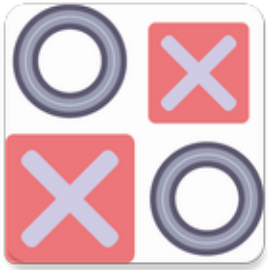
NGUYỄN NGỌC TOÀN MSSV: **44.01.104.197**

NGUYỄN TRẦN CAO PHONG MSSV: **44.01.104.169**

Lớp: **CNTT.C**

Báo cáo đồ án phát triển ứng dụng và giao diện

Cờ caro qua mạng LAN



Bình Chánh**, tháng** 11 **năm** 2020

Lời cảm ơn

Với những lời đầu tiên, chúng tôi xin được gửi đến ThS. Võ Tiến An – Giảng viên bộ môn Phát triển ứng dụng và giao diện Trường Đại học Sư Phạm Tp.HCM, cảm ơn thầy đã truyền đạt những kiến thức đầu tiên để làm tiền đề, cơ sở hoàn thành bài báo cáo này và đã, đang, sẽ trở thành những kinh nghiệm quý báu cho việc học tập và nghiên cứu của bọn tôi trong tương lai.

Đồng thời, chúng tôi cũng bày tỏ lòng biết ơn với quý Thầy Cô Trường Đại học Sư phạm Tp.HCM đã tận tình truyền dạy kiến thức trong quá trình học tập và thực hiện bài báo cáo này.

Cuối cùng, xin gửi lời cảm ơn tới những người thân trong gia đình và bạn bè luôn bên cạnh động viên, hỗ trợ về mặt tinh thần để chúng tôi vượt qua khó khăn và hoàn thiện bài báo cáo này.

Lời cam đoan

Chúng tôi xin cam đoan kết quả đạt được trong bài báo cáo là sản phẩm của riêng cá nhân, là kết quả của quá trình học tập và nghiên cứu khoa học độc lập. Trong toàn bộ nội dung của bài báo cáo, những điều được trình bày hoặc là của cá nhân hoặc là được tổng hợp từ nhiều nguồn tài liệu. Tất cả các nguồn tài liệu tham khảo đều có xuất xứ rõ ràng và được trích dẫn hợp pháp.

Chúng tôi xin hoàn toàn chịu trách nhiệm và các hình thức kỷ luật theo quy định cho lời cam đoan của mình.

Bình Chánh, tháng 11 năm 2020

Mục lục

MỞ ĐẦU

1. Lý do chọn đề tài

Với xu hướng phát triển mạnh mẽ của công nghệ cao thì cơ hội và nhu cầu việc làm của ngành Kỹ thuật phần mềm là rất lớn nên luôn thu hút đông đảo giới trẻ và luôn là ngành thời thượng trong lĩnh vực Thông tin và Truyền thông.

Nhằm đáp ứng yêu cầu của học phần và trang bị thêm các tri thức quý báu về lĩnh vực này. Mặt khác, để củng cố thêm kiến thức về mạng máy tính, việc kết nối và truyền, nhận các tín hiệu giữa các máy tính với nhau đã đặt ra bài toán thiết yếu cần phải giải và đó cũng chính là lý do đề tài này được lựa chọn.

2. Mục tiêu đề tài

Hoàn thiện được ứng dụng cờ Caro qua mạng LAN với các chức năng cơ bản sau:

+ *Kết nối và tạo server (Người dùng bấm button Connect đầu tiên).*
+ *Chức năng New game tạo mới lại bàn cờ ở cả hai máy khi có người dùng kích hoạt sự kiện New game.*
+ *Chức năng Undo (hoàn tác lại 2 nước đi trước đó).*
+ *Chức năng Quit.*
+ *Các thông báo khi kết thúc game, hết giờ, trò chơi kết thúc, thoát khỏi ứng dụng.*
+ *Tạo được gói setup để có thể cài đặt được ứng dụng trên các máy tính khác mà không cần source code.*
+ *Giao diện đơn giản, trực quan.*

3. Nội dung thực hiện

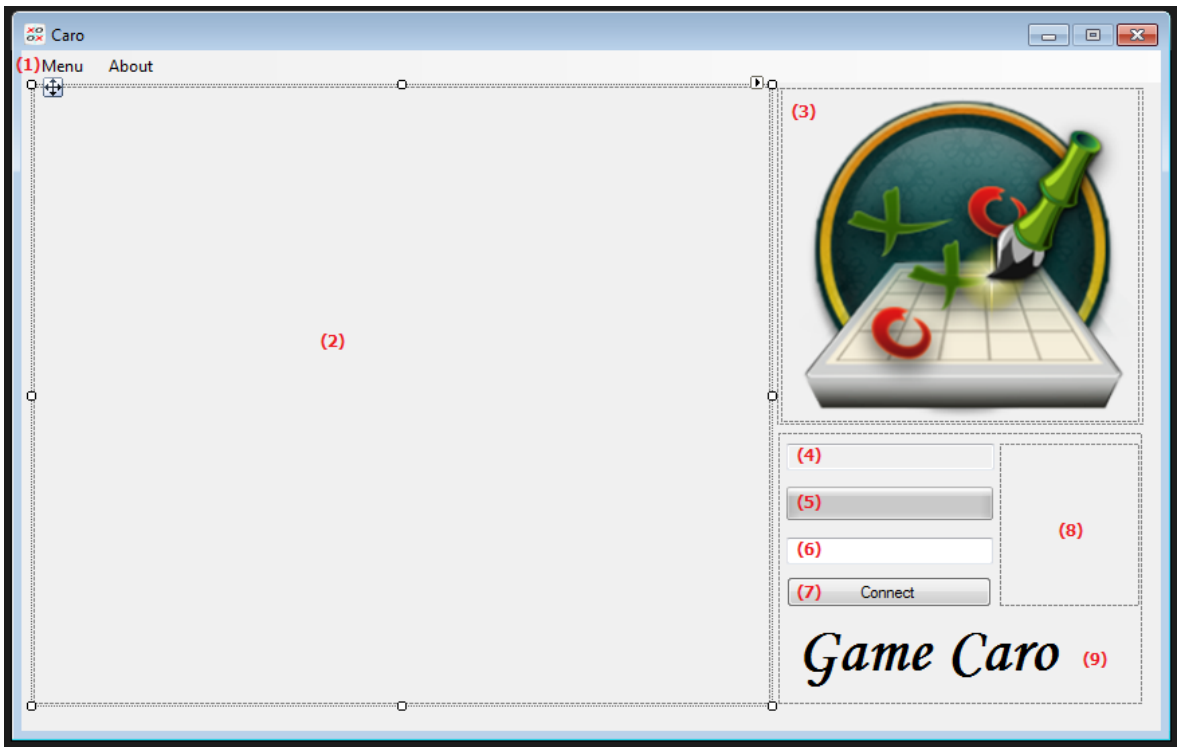
- *Tìm hiểu về nền tảng .NET và ngôn ngữ C# kết hợp với việc tham khảo các tài liệu do giảng viên cung cấp và các tài liệu mở trên Internet.*

- *Theo dõi seri Lập trình cờ Caro qua mạng LAN của chanel Howkteam tại đường dẫn: https://www.howkteam.vn/course/lap-trinh-game-caro-voi-c-winform-14*
- *Thực hành và ứng dụng các bài tập cơ bản mà giảng viên đã giao.*
- *Tiến hành lập trình, đóng gói và hoàn thiện ứng dụng như mục tiêu đã đề ra.*
- *Tổng kết và viết báo cáo*

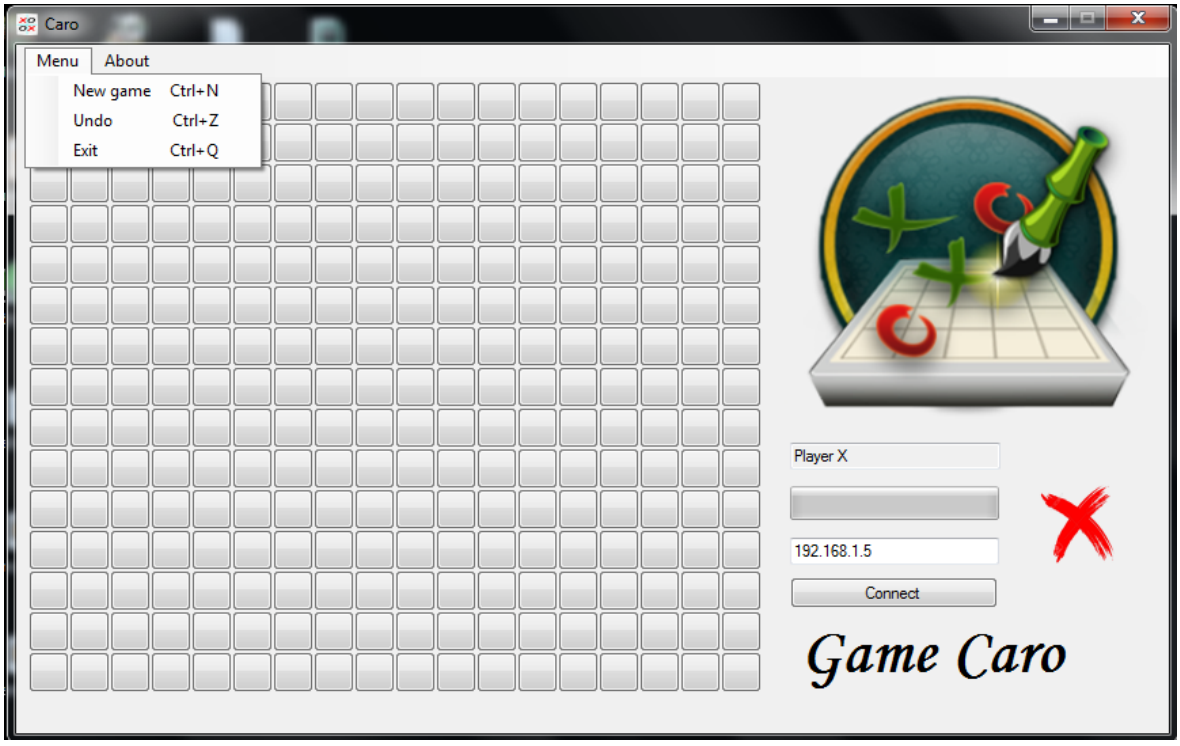
4. Phương pháp thực hiện

- *Sử dụng IDE Visual Studio 2013 để khởi tạo và bắt đầu quá trình lập trình ứng dụng.*
- *Áp dụng các kiến thức đã học tại trường, kết hợp với các kiến thức trên Internet và seri lập trình của Howkteam[1] để tiến hành hoàn thiện ứng dụng.*
- *Cài đặt và sử dụng Extension Visual Studio Installer để đóng gói ứng dụng thành file setup.exe.*
- *Sử dụng Version control github để có thể dễ dàng quản lý source code.*
- *Sử dụng office 365 để hoàn thiện báo cáo.*

I. Giao diện của chương trình:



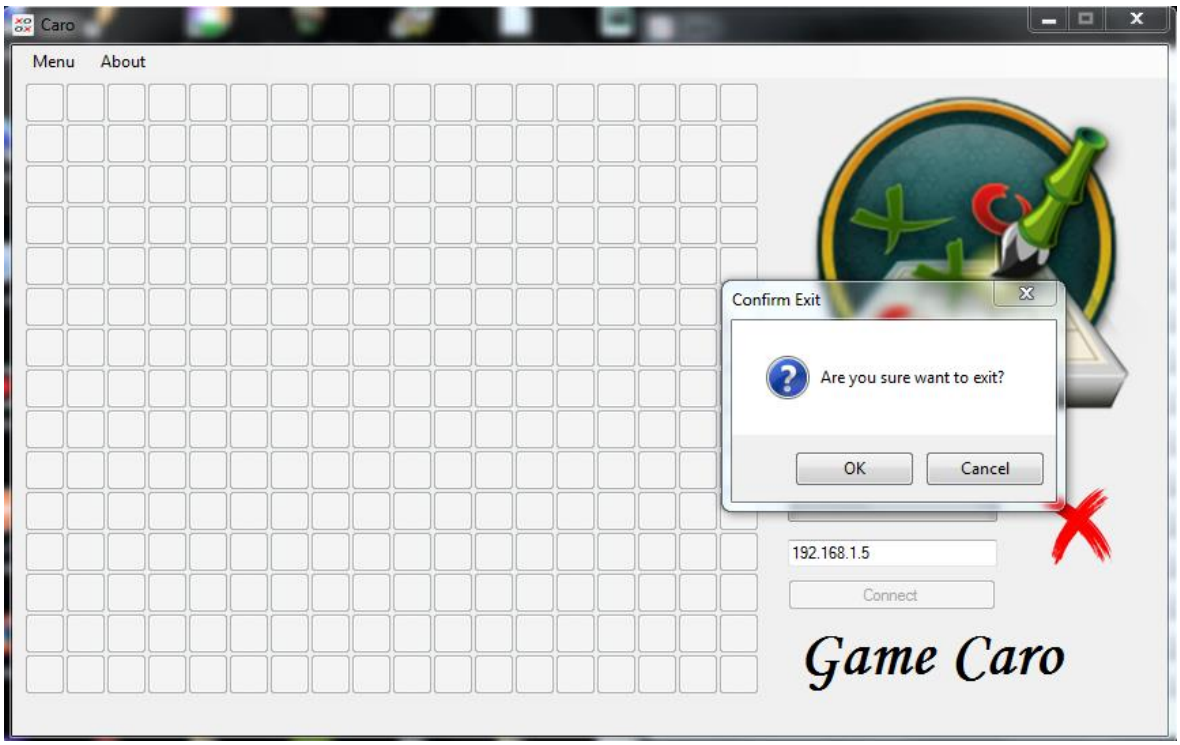
(1) Menu tool strip khi click sẽ xuất hiện các tùy chọn sau:

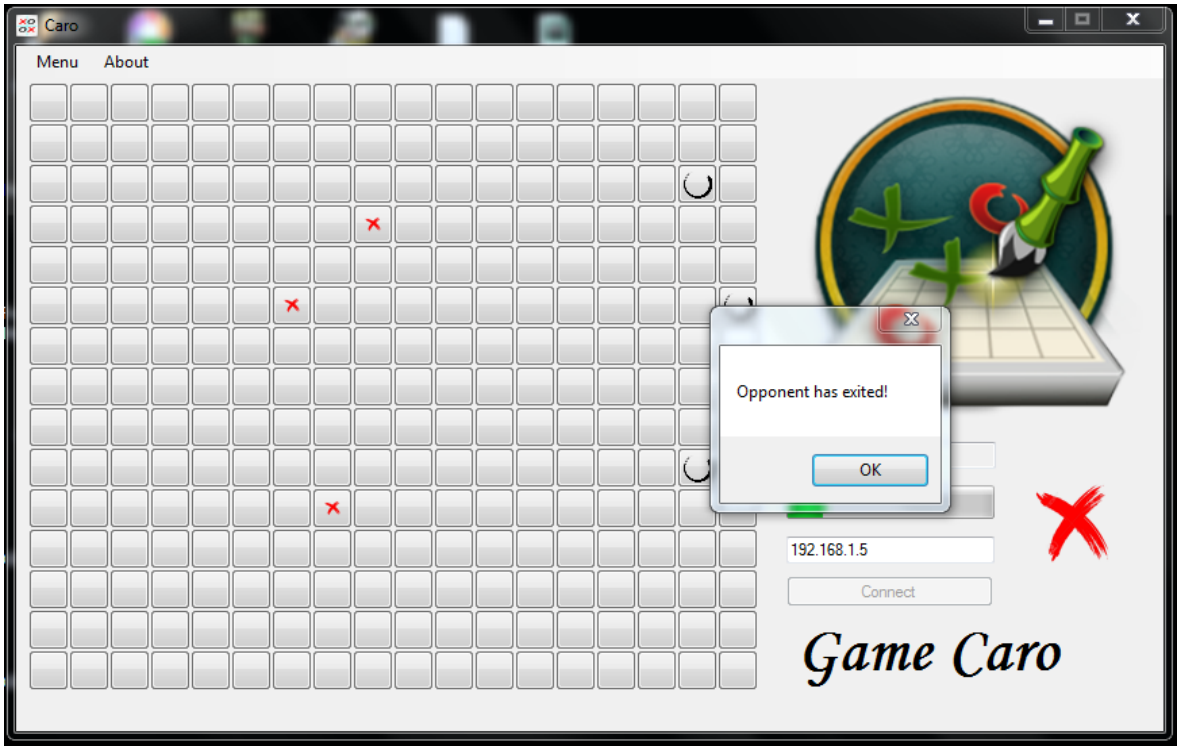


New game (Ctrl + N): Làm mới trò chơi ở cả hai máy khi có người chơi click vào hoặc nhấn Ctrl + N

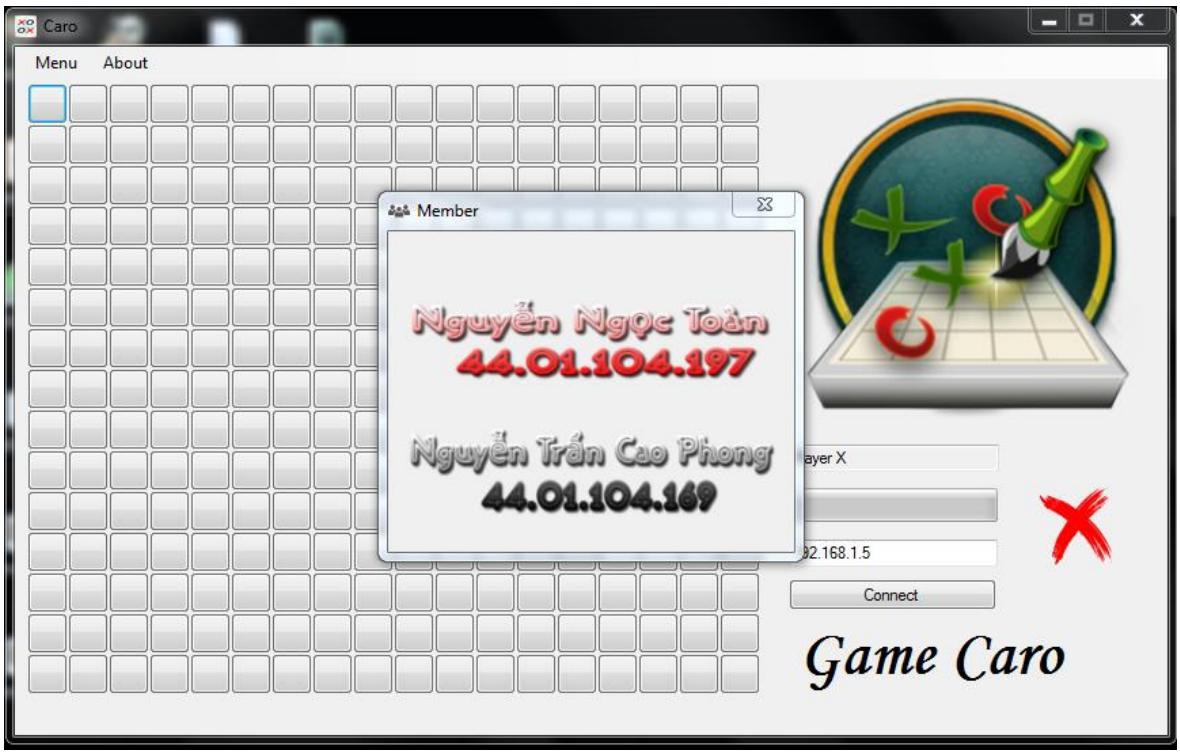
Undo (Ctrl + Z): Hoàn tác lại nước đi của bản thân người chơi và đối thủ khi click vào hoặc nhấn Ctrl + Z

Exit (Ctrl + Q): Hiện hộp thoại thông báo “Xác nhận thoát” nếu người chơi chọn Yes -> Thoát khỏi ứng dụng và thông báo đến người chơi còn lại.





Khi người dùng chọn About từ Menu tool strip sẽ xuất hiện hộp thoại danh sách các thành viên như sau:



(2) Panel dùng để tạo bàn cờ

(3) Panel chứa 1 picture box dùng để hiển thị logo trò chơi

(4) Text box dùng để hiển thị tên người chơi hiện tại (Bao gồm "Player X" và "Player O")

(5) Progress Bar dùng để đếm ngược thời gian trò chơi

(6) Text box hiển thị IP (Mặc định Ipv4 của mỗi máy tính đã cài đặt ứng dụng)

(7) Button dùng để kết nối LAN giữa hai máy với nhau

(8) Picture box hiển thị kí hiệu đại diện cho người chơi hiện tại

(9) Label hiển thị tên trò chơi

II. Các hằng số lưu trữ và kiểu dữ liệu tự định nghĩa:

Các hằng số được lưu trữ tại file Cons.cs nhằm có thể tái sử dụng dễ dàng, bao gồm:

```
namespace Caro
{
    public class Cons
    {
        public static int CHESS_WIDTH = 30;
        public static int CHESS_HEIGHT = 30;

        public static int CHESS_BOARD_WIDTH = 18;
        public static int CHESS_BOARD_HEIGHT = 15;

        public static int STEP_COOL_DOWN = 100;
        public static int TIME_COOL_DOWN = 10000;
        public static int INTERVAL_COOL_DOWN = 100;
    }
}
```

CHESS_WIDTH và **CHESS_HEIGHT**: lần lượt là hai giá trị của chiều rộng, chiều cao của mỗi một ô cờ trên bàn cờ.

CHESS_BOARD_WIDTH và **CHESS_BOARD_HEIGHT**: lần lượt là số ô cờ theo chiều rộng và chiều cao của bàn cờ.

STEP_COOL_DOWN và **TIME_COOL_DOWN**: lần lượt là các giá trị thuộc tính .Step và .Maximum của progress bar.

INTERVAL_COOL_DOWN: là giá trị của thuộc tính .Interval của Timer sẽ được đề cập sau.

Class Player được lưu trữ tại file Player.cs nhằm lưu được thông tin về người chơi bao gồm :Tên và kí hiệu đại diện cho người chơi

```
namespace Caro
{
    public class Player
    {
        private string name; // Ctrl + R + E

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        private Image mark;

        public Image Mark
        {
            get { return mark; }
            set { mark = value; }
        }

        public Player(string name, Image mark)
        {
            this.Name = name;
            this.Mark = mark;
        }
    }
}
```

Class PlayerInfo dùng để ghi lại tọa độ của button và người chơi hiện tại

```
namespace Caro
{
    public class PlayInfo
    {
        private Point point;

        public Point Point
        {
            get { return point; }
            set { point = value; }
        }

        private int currentPlayer;

        public int CurrentPlayer
        {
            get { return currentPlayer; }
            set { currentPlayer = value; }
        }

        public PlayInfo(Point point, int currentPlayer)
        {
            this.point = point;
            this.currentPlayer = currentPlayer;
        }
    }
}
```

Class SocketData dùng để lưu trữ tọa độ button, các thông điệp (message) và khối lệnh (command)

```
[Serializable]
public class SocketData
{
    private int command;


    public int Command
    {
        get { return command; }
        set { command = value; }
    }

    private Point point;

    public Point Point
    {
        get { return point; }
        set { point = value; }
    }

    private string message;

    public string Message
    {
        get { return message; }
        set { message = value; }
    }

    public SocketData(int command, string message, Point point)
    {
        this.Command = command;
        this.Message = message;
        this.Point = point;
    }
}
```

và có phương thức là một enum với các command sau:

```
public enum SocketCommand
{
    SEND_POINT,
    NOTIFY,
    NEW_GAME,
    UNDO,
    TIME_OUT,
    END_GAME,
    QUIT
}
```

nhằm giúp cho việc xử lý các chức năng giao tiếp giữa hai máy tính qua LAN và sẽ được đề cập sau.

Class SocketManager sẽ quy định các phương thức và thuộc tính về việc khởi tạo, truyền và nhận tín hiệu giữa hai máy tính qua kết nối LAN

Đối với Client:

```
public class SocketManager
{
    #region Client
    Socket client;
    public bool ConnectServer()
    {
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse(IP), PORT);
        client = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        try
        {
            client.Connect(iep);
            return true;
        }
        catch
        {
            return false;
        }

    }

    #endregion
```

Client sẽ được connect thông qua một IPEndPoint bằng phương thức IPEndPoint với hai tham số là IP và PORT sẽ được đề cập sau.

Đối với Server:

```
    #region Server

    Socket server;
    public void CreateServer()
    {
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse(IP), PORT);
        server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

        server.Bind(iep);
        server.Listen(10);

        Thread acceptClient = new Thread(() =>
        {
            client = server.Accept();
        });
        acceptClient.IsBackground = true;
        acceptClient.Start();
    }

    #endregion
```

Tương tự đối với server, nhưng lúc này sẽ thêm các phương thức Bind và Listen nhằm đợi tín hiệu kết nối từ Client. Ngoài ra còn có thêm luồng (Thread) khác nhằm không làm ảnh hưởng đến hiệu suất của chương trình.

```
#region Both
public string IP = "127.0.0.1";
public int PORT = 9999;
public bool isServer = true;
public const int BUFFER = 1024;
```

Sau đây là IP và PORT đã đề cập đến, tiếp đến là một biến bool isServer nhằm phân biệt giữa Client và Server, BUFFER quy định độ lớn của kiểu dữ liệu mỗi lần truyền và nhận.

```
public bool Send(object data)
{
    byte[] sendData = SerializeData(data);

    return SendData(client, sendData);

}

public object Receive()
{
    byte[] receiveData = new byte[BUFFER];
    bool isOK = ReceiveData(client, receiveData);

    return DeserializeData(receiveData);
}
```

Phương thức Send và Receive lần lượt tương ứng với việc gửi và nhận dữ liệu dưới dạng byte[]

```
// Nén đối tượng thành mảng byte
public byte[] SerializeData(Object o)
{
    MemoryStream ms = new MemoryStream();
    BinaryFormatter bf1 = new BinaryFormatter();
    bf1.Serialize(ms, o);
    return ms.ToArray();
}

// Giải nén 1 mảng byte thành 1 đối tượng (object)
public object DeserializeData(byte[] theByteArray)
{
    MemoryStream ms = new MemoryStream(theByteArray);
    BinaryFormatter bf1 = new BinaryFormatter();
    ms.Position = 0;
    return bf1.Deserialize(ms);
}
```

```
// Lấy ra IPv4 của card mạng đang dùng
public string GetLocalIPv4(NetworkInterfaceType _type)
{
    string output = "";
    foreach (NetworkInterface item in NetworkInterface.GetAllNetworkInterfaces())
    {
        if (item.NetworkInterfaceType == _type && item.OperationalStatus == OperationalStatus.Up)
        {
            foreach (UnicastIPAddressInformation ip in item.GetIPProperties().UnicastAddresses)
            {
                if (ip.Address.AddressFamily == AddressFamily.InterNetwork)
                {
                    output = ip.Address.ToString();
                }
            }
        }
    }
    return output;
}
```

và các hàm mở rộng như trên để hoàn thiện các tính năng thiết yếu giữa việc kết nối LAN.

III. Khởi tạo bàn cờ:

Bàn cờ trò chơi sẽ được khởi tạo thông qua hàm sau:

```
void DrawChessBoard()
{
    pnlChessBoard.Enabled = true;
    pnlChessBoard.Controls.Clear();
    PlayTimeLine = new Stack<PlayInfo>();

    Matrix = new List<List<Button>>();
    this.Player = new List<Player>() {
        new Player ("Player X", Image.FromFile(Application.StartupPath + "\\assets\\x_symbol.png")),
        new Player ("Player O", Image.FromFile(Application.StartupPath + "\\assets\\o_symbol.png"))
    };
    CurrentPlayer = 0;
    ChangePlayer();
    Button firstButton = new Button()
    {
        Width = 0, Location = new Point(0, 0)
    };
    for (int i = 0; i < Cons.CHESS_BOARD_HEIGHT; i++)
    {
        Matrix.Add(new List<Button>());
        for (int j = 0; j < Cons.CHESS_BOARD_WIDTH; j++)
        {
            Button btn = new Button(){
                Width = Cons.CHESS_WIDTH,
                Height = Cons.CHESS_HEIGHT,
                Location = new Point(firstButton.Location.X + firstButton.Width, firstButton.Location.Y),
                BackgroundImageLayout = ImageLayout.Stretch,
                Tag = i.ToString(),
            };
            btn.Click += btn_Click;
            pnlChessBoard.Controls.Add(btn);
            Matrix[i].Add(btn);
            firstButton = btn;
        }
        firstButton.Location = new Point(0, firstButton.Location.Y + Cons.CHESS_HEIGHT);
        firstButton.Width = 0;
        firstButton.Height = 0;
    }
}
```

Đầu tiên, khi chương trình được khởi chạy sẽ gọi hàm DrawChessBoard để có thể vẽ bàn cờ. Panel chứa các button sẽ được Enable, đồng thời sẽ xóa hết các dữ kiện trước đó (Do khi ta gọi hàm New game thì phải vẽ lại panel đồng thời xóa hết các dữ kiện mà người chơi đã thực hiện trước đó *nếu có).

Khởi tạo một Stack thuộc Class PlayInfo nhằm lưu lại các thông tin về trò chơi để có thể thực hiện các xử lý về sau.

Khởi tạo một kiểu dữ liệu Matrix List lồng List tương đương với mảng hai chiều trong các ngôn ngữ khác nhằm lưu lại tọa độ của Button để có thể xử lý trong tương lai như Class PlayInfo đã nêu trên.

Ta khởi tạo danh sách người chơi Player gồm hai phần tử tương đương với X (có chỉ số là 0) và O (có chỉ số là 1) cùng ký hiệu đại diện tương ứng. Biến CurrentPlayer sẽ mang giá trị ban đầu là 0 tương ứng với X.

Kế đến sẽ thông qua hai vòng lặp For để có thể tạo nên bàn cờ dựa trên các hằng số đã quy ước ở File Cons.cs đồng thời thông qua thuộc tính Tag để lưu lại tọa độ của button sau đó truyền vào trong kiểu dữ liệu Matrix đã khởi tạo trước đó.

IV. Xử lý đổi người chơi và kết quả của trò chơi:

Việc chuyển đổi qua lại giữa hai người chơi được thực hiện qua sự kiện click của bất kì button nào thuộc panel bàn cờ.

```
void btn_Click(object sender, EventArgs e)
{
    Button btn = sender as Button;
    if (btn.BackgroundImage != null)
        return;
    Mark(btn);

    PlayTimeLine.Push(new PlayInfo(GetChessPoint(btn), CurrentPlayer));

    CurrentPlayer = CurrentPlayer == 1 ? 0 : 1;

    ChangePlayer();

    tmCooldown.Start();
    pnlChessBoard.Enabled = false;
    prcbTime.Value = 0;

    socket.Send(new SocketData((int)SocketCommand.SEND_POINT, "", GetChessPoint(btn)));
    undoToolStripMenuItem.Enabled = false;

    Listen();

    if (isEndGame(btn))
    {
        EndGame();
        socket.Send(new SocketData((int)SocketCommand.END_GAME, "", new Point()));
    }
}
```

Bằng việc kết hợp giữa biến CurrentPlayer nhằm chuyển đổi tuần tự qua lại giữa người chơi X và người chơi O cùng với hàm Mark (dùng để đánh dấu người chơi) và hàm ChangePlayer (thay đổi tên người chơi tại text box hiển thị và kí hiệu đại diện cho người chơi đó ở picture box đã đề cập ở trên).

Việc xử lý kết quả của trò chơi sẽ được thông qua hàm isEndGame sau:

```
private bool isEndGame(Button btn)
{
    return isEndHorizontal(btn) || isEndVertical(btn) || isEndMainDiagonal(btn) || isEndSecondaryDiagonal(btn);
}
```

Hàm này sẽ trả về true khi thỏa mãn một trong bốn hàm kế tiếp:

```
private bool isEndHorizontal(Button btn)
{
    Point point = GetChessPoint(btn);

    int countLeft = 0;
    for(int i = point.X; i >= 0; i--){
        if (Matrix[point.Y][i].BackgroundImage == btn.BackgroundImage)
        {
            countLeft++;
        }
        else
            break;
    }

    int countRight = 0;
    for (int i = point.X + 1; i <Cons.CHESS_BOARD_WIDTH; i++)
    {
        if (Matrix[point.Y][i].BackgroundImage == btn.BackgroundImage)
        {
            countRight++;
        }
        else
            break;
    }

    return countLeft + countRight == 5;
}
```

```
private bool isEndVertical(Button btn)
{
    Point point = GetChessPoint(btn);

    int countUp = 0;
    for (int i = point.Y; i >= 0; i--)
    {
        if (Matrix[i][point.X].BackgroundImage == btn.BackgroundImage)
        {
            countUp++;
        }
        else
            break;
    }

    int countDown = 0;
    for (int i = point.Y + 1; i < Cons.CHESS_BOARD_HEIGHT; i++)
    {
        if (Matrix[i][point.X].BackgroundImage == btn.BackgroundImage)
        {
            countDown++;
        }
        else
            break;
    }

    return countUp + countDown == 5;
}
```

```
private bool isEndMainDiagonal(Button btn)
{
    Point point = GetChessPoint(btn);

    int countUp = 0;
    for (int i = 0; i <= point.X; i++)
    {
        if (point.X - i < 0 || point.Y - i < 0)
            break;
        if (Matrix[point.Y - i][point.X - i].BackgroundImage == btn.BackgroundImage)
        {
            countUp++;
        }
        else
            break;
    }

    int countDown = 0;
    for (int i = 1; i <= Cons.CHESS_BOARD_WIDTH - point.X; i++)
    {
        if (point.Y + i >= Cons.CHESS_BOARD_HEIGHT || point.X + i >= Cons.CHESS_BOARD_WIDTH)
            break;
        if (Matrix[point.Y + i][point.X + i].BackgroundImage == btn.BackgroundImage)
        {
            countDown++;
        }
        else
            break;
    }

    return countUp + countDown == 5;
}
```

```
  private bool isEndSecondaryDiagonal(Button btn)
  {
      Point point = GetChessPoint(btn);

      int countUp = 0;
      for (int i = 0; i <= point.X; i++)
      {
          if (point.X + i > Cons.CHESS_BOARD_WIDTH || point.Y - i < 0)
              break;
          if (Matrix[point.Y - i][point.X + i].BackgroundImage == btn.BackgroundImage)
          {
              countUp++;
          }
          else
              break;
      }
      int countDown = 0;
      for (int i = 1; i <= Cons.CHESS_BOARD_WIDTH - point.X; i++)
      {
          if (point.Y + i >= Cons.CHESS_BOARD_HEIGHT || point.X - i < 0)
              break;
          if (Matrix[point.Y + i][point.X - i].BackgroundImage == btn.BackgroundImage)
          {
              countDown++;
          }
          else
              break;
      }
      return countUp + countDown == 5;
  }
```

Lần lượt là các trường hợp chiến thắng theo hàng ngang, dọc, đường chéo chính và chéo phụ. Ý tưởng đơn giản như sau, từ button đang xét lần lượt xét số lượng số button ở phải và trái (đối với trường hợp ngang, dọc), số lượng button chéo trên và chéo dưới (đối với trường hợp hai đường chéo) nếu tổng số lượng bằng 5 thì trả về true. Việc xét tọa độ của button như thế dựa trên hàm xét tọa độ

```
private Point GetChessPoint(Button btn)
{
    int vertical = Convert.ToInt32(btn.Tag);
    int horizontal = Matrix[vertical].IndexOf(btn);
    Point point = new Point(horizontal, vertical);
    return point;
}
```

sẽ trả về kết quả là một Point với tọa độ ta đã lưu trước đó vào biến Matrix kiểu dữ liệu List lồng List đã khai báo trên.

V. Xử lý đếm ngược thời gian:

Việc đếm ngược thời gian sẽ được thông qua progress bar để hiển thị ở giao diện và một luồng riêng là Timer để không làm ảnh hưởng đến hiệu xuất của chương trình.

```
    prcbTime.Step = Cons.STEP_COOL_DOWN;
    prcbTime.Maximum = Cons.TIME_COOL_DOWN;
    prcbTime.Value = 0;

    tmCooldown.Interval = Cons.INTERVAL_COOL_DOWN;
```

Thông qua event Tick của timer

```
private void tmCooldown_Tick(object sender, EventArgs e)
{
    prcbTime.PerformStep();

    if (prcbTime.Value >= prcbTime.Maximum)
    {
        EndGame();
        socket.Send(new SocketData((int)SocketCommand.TIME_OUT, "", new Point()));
    }
}
```

khi giá trị của Progress bar >= giá trị tối đa Maximum sẽ gọi hàm EndGame để kết thúc trò chơi (Trò chơi sẽ kết thúc theo trường hợp Time out – hết giờ, việc xử lý này sẽ được đề cập ở phần IX). Đồng thời mỗi khi trò chơi được New game hoặc kết thúc thì timer sẽ được stop. Ngoài ra, khi bất kì một button nào thuộc panel bàn cờ được click hay chức năng undo được kích hoạt thì progress bar sẽ được reset nhằm đảm bảo tính logic.

VI. Chức năng Undo:

Về việc undo (hoàn tác các nước đi trước đó) sẽ thông qua các hàm sau:

```
private bool UndoAStep()
{
    prcbTime.Value = 0;
    if (PlayTimeLine.Count <= 0)
    {
        prcbTime.Value = 0;
    }
    PlayInfo oldPlayer = PlayTimeLine.Pop();
    Button btn = Matrix[oldPlayer.Point.Y][oldPlayer.Point.X];

    btn.BackgroundImage = null;

    if (PlayTimeLine.Count <= 0)
    {
        CurrentPlayer = 0;
    }
    else
    {
        oldPlayer = PlayTimeLine.Peek();
    }

    ChangePlayer();

    return true;
}
```

Do ta cần undo cả hai nước đi một lần nhằm đảm bảo tính công bằng cho trò chơi, trước tiên ta sẽ xét hàm undo một nước. Về việc này ta sẽ dựa trên Stack PlayInfo đã khai báo trước đó và phương thức Pop nhằm lấy ra được người chơi gần đây nhất thông qua PlayTimeLine. Qua đó ta sẽ có được tọa độ của nước đi gần đây nhất, kế đến reset background của button về null. Nếu Stack rỗng thì gán giá trị người chơi hiện tại là 0 (tương đương với X hay nói cách khác, giá trị mặc định lúc bắt đầu trò chơi), nếu Stack vẫn chưa rỗng thì xem

xét giá trị kế đó thông qua phương thức Peek (Không loại bỏ phần tử khỏi Stack như phương thức Pop) và cuối cùng gọi hàm ChangePlayer để hoán đổi người chơi.

Do đã đề cập ở trên, nhằm đảm bảo tính công bằng cho trò chơi, ta sẽ tiến hành Undo hai nước một lần, việc này sẽ được thực hiện thông qua hàm:

```
bool Undo()
{
    try
    {
        if (PlayTimeLine.Count <= 0)
        {
            return false;
        }

        bool isUndo1 = UndoAStep();
        bool isUndo2 = UndoAStep();

        PlayInfo oldPlayer = PlayTimeLine.Peek();
        CurrentPlayer = oldPlayer.CurrentPlayer == 1 ? 0 : 1;

        return isUndo1 && isUndo2;
    }
    catch
    {
        return false;
    }
}
```

Tương tự như hàm UndoAStep ở trên, ta sẽ gọi hai lần hàm này để có thể Undo được hai lần như ý muốn đồng thời ta sẽ hoán đổi lại chỉ số người chơi hiện tại tuần tự dựa trên người chơi trước đó.

VII. Xử lý kết nối LAN giữa hai máy:

Việc kết nối giữa hai máy thông qua LAN sẽ được thực hiện thông qua các Class SocketData và SocketManager đã nêu ở trên đồng thời là hàm:

```
private void ProcessData(SocketData data)
{
    switch (data.Command)
    {
        case (int)SocketCommand.NEW_GAME:
            this.Invoke((MethodInvoker)(() =>
            {
                NewGame();
                pnlChessBoard.Enabled = false;
            }));
            break;
        case (int)SocketCommand.QUIT:
            tmCooldown.Stop();
            MessageBox.Show(this, "Opponent has exited!");
            break;
        case (int)SocketCommand.SEND_POINT:
            this.Invoke((MethodInvoker)(() =>
            {
                prcbTime.Value = 0;
                pnlChessBoard.Enabled = true;
                tmCooldown.Start();
                OtherPlayerMark(data.Point);
                undoToolStripMenuItem.Enabled = true;
            }));
            break;
        case (int)SocketCommand.UNDO:
            Undo();
            prcbTime.Value = 0;
            break;
        case (int)SocketCommand.TIME_OUT:
            MessageBox.Show(this, "Time out!");
            break;
        case (int)SocketCommand.END_GAME:
            MessageBox.Show(this, "Game over!!");
            break;
        default:
            break;
    }
    Listen();
}
```

Socket đã được khai báo ngay từ lúc khởi tạo chương trình

```
socket = new SocketManager();
```

đồng thời được sử dụng mỗi khi cần truyền đi các tín hiệu như:

```
socket.Send(new SocketData((int)SocketCommand.SEND_POINT, "", GetChessPoint(btn)));
```

```
socket.Send(new SocketData((int)SocketCommand.END_GAME, "", new Point()));
```

```
socket.Send(new SocketData((int)SocketCommand.TIME_OUT, "", new Point()));
```

```
socket.Send(new SocketData((int)SocketCommand.NEW_GAME, "", new Point()));
```

```
socket.Send(new SocketData((int)SocketCommand.UNDO, "", new Point()));
```

```
socket.Send(new SocketData((int)SocketCommand.QUIT, "", new Point()));
```

tương ứng với các event của chương trình: New game, Send point (mỗi khi người chơi tiến hành đánh cờ), end game (kết thúc khi có người chiến thắng), time out (kết thúc khi hết thời gian quy định), undo (hoàn tác), quit (khi có người chơi thoát).

```
private void Form1_Shown(object sender, EventArgs e)
{
    txbIP.Text = socket.GetLocalIPv4(NetworkInterfaceType.Wireless80211);

    if (string.IsNullOrEmpty(txbIP.Text))
    {
        txbIP.Text = socket.GetLocalIPv4(NetworkInterfaceType.Ethernet);
    }
}
void Listen()
{
        Thread listenThread = new Thread(() =>
        {
            try
            {
                SocketData data = (SocketData)socket.Receive();
                ProcessData(data);
            }
            catch { }
        });
        listenThread.IsBackground = true;
        listenThread.Start();
}
```

Bên cạnh đó, việc này dựa trên event Shown của Form_1, khi bắt đầu chương trình, sẽ tiến hành lấy Ipv4 của từng máy tính để hiện thị tại text box IP như đã nêu trước đó. Đồng thời hàm Listen dùng để tiếp nhận các tín hiệu giữa server và client sẽ được áp dụng một luồng Thread riêng nhằm tránh xung đột và giảm đi hiệu xuất của chương trình.

VIII. Link repo và tài liệu tham khảo:

Link repo: https://github.com/darkhunterLearning/PTGDUD

[1] https://www.howkteam.vn/course/lap-trinh-game-caro-voi-c-winform-14

[2] Các slide bài giảng của thầy Võ Tiến An – giảng viên bộ môn Phát triển ứng dụng và giao diện trường ĐH Sư phạm TP.HCM