BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA/VIỆN: CÔNG NGHỆ THÔNG TIN**

**---****---**



**BÁO CÁO THỰC TẬP CƠ SỞ**

**CHƯƠNG TRÌNH CHƠI CỜ CARO 2 NGƯỜI CHƠI THEO PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

|  |  |  |
| --- | --- | --- |
| **Giảng viên hướng dẫn** | **:** | **Nguyễn Đình Hưng** |
| **Sinh viên thực hiện** | **:** | **Nguyễn Quốc Châu** |
| **Mã số sinh viên** | **:** | **61130073** |

# Giới thiệu

Game Caro làm một trò chơi rất phổ biến nhưng nó thường được chơi trên giấy vở và rất tốn nhiều giấy mực. Vì thế để giải quyết vấn đề giấy mực em xây dựng phần mềm Game Caro cơ bản 2 người chơi trên giao diện WinForm của Visual Studio để người chơi có trải nghiệm tốt hơn.

# Phân tích bài toán

* Tạo bàn cờ (Board Chess).
* Kiểm soát những ô đã được đánh (hình ảnh O,X,…).
* Kiểm soát lượt đánh của người chơi (chỉ được đánh 1 lần).
* Hỗ trợ đánh lại. Một lần Undo quay lại 2 nước (Undo).
* Tính thời gian đánh của mỗi người chơi.
* Tính năng tạo game mới (New Game), thoát game (Quit).
* Kiểm tra chiến thắng (kiểm tra đường chéo chính,chéo phụ, hàng ngang và hàng dọc qua luật lệ >5 điểm thẳng hàng là thắng).
* Kết nối 2 người chơi thông qua mạng LAN (kết nối Client đến Server thông qua IPV4).

# Tìm hiểu giải thuật

* Tạo Class Contants chứa các biến hằng.
* Tạo Class Player chứa các thuộc tính tên,hình ảnh, kí tự của người chơi.
* Tạo Class PlayInfo chứa các thuộc tính tọa độ,trạng thái, kí tự của người chơi.
* Tạo Class GameBoard
* Khởi tạo 6 thuộc tính
* Timer TimerStop (xuất ra form giao diện).
* Panel ChessBoard (xuất ra form giao diện).
* PictureBox Avatar (xuất ra form giao diện).
* TextBox PlayerName (xuất ra form giao diện).
* Int CurrentPlayer (nhận trạng thái người đánh).
* List<List<Button>> MatrixPositions (lưu trữ danh sách các button đã được đánh theo kiểu mảng).
* Stack<PlayInfo> PlayTimeLine (tạo ngăn xếp chứa các button đã được đánh thông qua push).
* List<Player> players (Tag dùng chứa danh sách người chơi thông qua gọi Class Player).
* event EventHandler<ButtonClickEvent> PlayerMarked (lắng nghe sự kiện khi người chơi click).
* Hàm DrawGameBoard() vẽ bàn cờ bằng cách dùng 2 vòng lặp lồng nhau. In các button ra form
* Hàm Symbol() giúp in ra kí tự người chơi vào button khi được đánh vào và đổi trạng thái CurrentPlayer sau mỗi lượt đánh.
* Hàm ChangePlayer() giúp in ra tên người chơi và hình ảnh người chơi.
* Hàm Btn\_Click() xử lý sự kiện button đánh. Nếu button đó được đánh rồi thì sẽ không được đánh tiếp (if (btn.BackgroundImage != null) return;) và khi được đánh sẽ lưu sự kiện đánh button đó (if (playerMarked!=null)playerMarked(this,new ButtonClickEvent(GetChessPoint(btn)));) . Xử lý sự kiện kết thúc game if (isEndGame(btn))EndGame();
* Hàm OtherPlayerClicked() giúp xử lý sự kiện đánh 2 người, tương tự như hàm Btn\_Click ta chỉ thêm Button btn = MatrixPositions [point.Y][point.X]; giúp xử lý được vị trí đánh từ 2 form
* Hàm EndGame() sẽ cho Timer form dừng, bàn cờ sẽ không được đánh (= false). Và vẽ lại bàn cờ gọi lại lại DrawGameBoard().
* Hàm bool isEndGame() nhận giá trị button của người đánh trả về 4 trường hợp xử thắng là isEndRow(button) là xử lý hàng ngang. isEndCol(button) là xử lý hàng dọc. isEndMainDiagonal(button) là xử lý đường chéo chính. isEndAuxiliaryDiagonal(button) là xử lý đường chép phụ.
* Hàm Point GetChessPoint() là hàm xử lý mảng chứa các button được tạo trước đó để lấy ra các tọa độ để dễ dàng xử lý thắng.
* int vertical = Convert.ToInt32(button.Tag); lấy ra tọa độ là số của button theo chiều dọc.
* int horizontal = MatrixPositions[vertical].IndexOf(button); lấy ra tọa độ là số của button theo chiều ngang.
* Khởi tạo đối tượng point chứa vertical, horizontal và return point.
* Hàm bool isEndRow() ta lấy lại thuộc tính point ở hàm GetChessPoint() dùng. Tạo biến countLeft=0, countRight=0.
* Vòng lặp for duyệt từ vị trí X đến tận cùng bàn cờ là bên trái. Kiểm tra ký hiệu có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countLeft++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Vòng lặp for duyệt từ vị trí X+1 đến tận cùng bàn cờ là bên phải. Kiểm tra ký hiệu có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countLeft++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Nếu countLeft + countRight >=5 thì return là true và ngược lại.
* Hàm bool isEndCol() ta lấy lại thuộc tính point ở hàm GetChessPoint() dùng. Tạo biến countTop=0, countBottom=0.
* Vòng lặp for duyệt từ vị trí Y đến tận cùng bàn cờ là phía trên. Kiểm tra ký hiệu có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countTop++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Vòng lặp for duyệt từ vị trí Y+1 đến tận cùng bàn cờ là phía dưới. Kiểm tra ký hiệu có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countBottom++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Nếu countTop + countBottom >=5 thì return là true và ngược lại.
* Hàm bool isEndMainDiagonal() ta lấy lại thuộc tính point ở hàm GetChessPoint() dùng. Tạo biến countTop=0, countBottom=0.
* Vòng lặp for duyệt từ vị trí 0 đến vị trí X (duyệt trên xuống đến vị trí X). Kiểm tra ký hiệu (từ vị trí 0:0 trở đi) có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countTop++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Vòng lặp for duyệt từ vị trí X+1 đến tận cùng chiều dài bàn cờ. Kiểm tra ký hiệu (từ vị trí CellWidth:CellHeight duyệt lên vị trí X) có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countBottom++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Nếu countTop + countBottom >=5 thì return là true và ngược lại.
* Hàm bool isEndAuxiliaryDiagonal() ta lấy lại thuộc tính point ở hàm GetChessPoint() dùng. Tạo biến countTop=0, countBottom=0.
* Vòng lặp for duyệt từ vị trí 0 đến vị trí X (duyệt trên xuống đến vị trí X). Kiểm tra ký hiệu (từ vị trí CellWidth:CellHeight duyệt xuống vị trí X) có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countTop++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Vòng lặp for duyệt từ vị trí X+1 đến tận cùng chiều dài bàn cờ. Kiểm tra ký hiệu (từ vị trí CellWidth:CellHeight duyệt lên vị trí X) có cùng với ký hiệu của người chơi hiện đang đánh thì tăng biến đếm countBottom++. Khi có một ký tự không giống ký tự người chơi thì sẽ break khỏi vòng lặp.
* Nếu countTop + countBottom >=5 thì return là true và ngược lại.
* Hàm bool UndoAsStep() khởi tạo đối tượng PlayInfo oldPoint = PlayTimeLine.Pop();. Dùng Pop() là lấy hẳn một đối tượng ra khỏi Stack. Sau khi lấy ra cho button có BackgroundImage là null. Gọi lại hàm ChangePlayer() để cập nhật lại ảnh và tên người chơi trên form.
* Hàm bool Undo() gọi lại hàm UndoAsStep() lặp lại 2 lần. Giúp xác định dễ được CurrentPlayer đánh tiếp theo.
* Tạo Class SocketManager
* Tạo Socket client, Socket server để sử dụng lớp Socket.
* Hàm ConnectServer() Tạo đối tượng lấy IP và Port. Khởi tạo client lấy InterNetwork (dùng chung một mạng), sử dụng giao thức Protocols. Client kết nối đúng IP và Port sẽ trả về giá trị true.
* Hàm CreateServer() Tạo đối tượng lấy IP và Port. Khởi tạo server lấy InterNetwork (dùng chung một mạng), sử dụng giao thức Protocols. Server sẽ lắng nghe thông qua hàm Listen(). Tạo Thread để tạo luồng nhận kết nối client.
* Hàm SerializeData() nén đối tượng thành mảng byte[]
* Hàm DeserializeData() giải nén mảng byte[] thành đối tượng object
* Hàm SendData() sẽ gửi một mảng byte[]. Return Send(data).
* Hàm ReceiveData() sẽ nhận một mảng byte[]. Return Receive(data).
* Hàm Send() khởi tạo đối tượng byte[] sendData =SerializeData(data);. Nhận mảng byte và return SendData(client, sendData);
* Hàm Receive() khởi tạo đối tượng byte[] receiveData=SerializeData( data). Nhận mảng byte và return DeserializeData(receiveData);
* Hàm GetLocalIPv4() lấy IPV4 của hệ thống để hiển thị lên form dễ dàng cho client kết nối.
* Tạo Class SocketData
* Khởi tạo 3 thuộc tính
* Command nhận các lệnh để xử lý data
* Point lệnh gửi vị trí đánh của người chơi
* Message
* Hàm liệt kê enum SocketCommand chứa các hành động khi chơi game như: Send\_point, new\_game, quit, ….
* Hàm btn\_Lan\_Click() xử lý sự kiện kết nối LAN, form nào ấn kết nối trước sẽ là server và ngược lại là client.
* Hàm form\_shown() hiện IPV4 của máy lên trên form.
* Hàm Listen() lắng nghe hành động của form và xử lý dữ liệu thông qua hàm ProcessData().
* Hàm ProcessData() xử lý các command của socket và tiếp tục lắng nghe thông qua hàm Listen() để có sự trao đổi tiên tục giữa hai form người chơi.
* Hàm ChessBoard\_PlayerMarked() xử lý sự kiện khi người chơi click vào button sẽ được xử lý.
* Hàm xử lý hết thời gian tmCoolDown\_Tick() xử lý khi thanh ProgressBar chạy hết thời gian Timer thì kết thúc game
* Hàm Form\_KeyDown() xử lý phím tắt nhanh khi chơi game.
* Tạo các hàm EndGame(),NewGame(),Quit() xử lý các tính năng của form.

# Tìm hiểu công cụ lập trình

Visual studio là một trong những công cụ hỗ trợ lập trình website rất nổi tiếng nhất hiện nay của Mcrosoft và chưa có một phần mềm nào có thể thay thế được nó. Visual Studio được viết bằng 2 ngôn ngữ đó chính là C# và VB+. Đây là 2 ngôn ngữ lập trình giúp người dùng có thể lập trình được hệ thống một các dễ dàng và nhanh chóng nhất thông qua Visual Studio. Được đánh giá cao như vậy bởi Visual Studio sở hữu nhiều tính năng cực kỳ hấp dẫn. Cụ thể: Đa nền tảng, Đa ngôn ngữ lập trình, Hỗ trợ website, Kho tiện ích mở rộng phong phú, Lưu trữ phân cấp, Kho lưu trữ an toàn, Màn hình đa nhiệm, Hỗ trợ viết code, Hỗ trợ thiết bị đầu cuối, Hỗ trợ Git, Intellisense, Tính năng comment.

Khả năng tạo Class riêng biệt dễ dàng sửa lỗi thông qua trình Debug. Hỗ trợ các hàm xử lý nhanh chóng . Có tính đóng gói giúp lập trình viên có thể tận dụng Tính đóng gói khi muốn bảo vệ dữ liệu bên trong của đối tượng. Dữ liệu đó hoàn toàn không thể bị sửa đổi một cách bất ngờ bởi những mã lệnh bên ngoài từ những phần khác của chương trình.

Hỗ trợ giao diện WinForm dễ dàng tạo và thiết lập . Phù hợp với tạo các phần mềm như: quản lý sinh viên, quản lý hàng hóa,….

# Mô tả giải thuật

* Class Contant
* 2 thuộc tính độ rộng,cao của 1 ô button trên bàn cờ.
* CellWidth = 35;
* CellHeight = 35;
* 2 thuộc tính độ rộng, cao của bàn cờ dùng để vẽ bàn cờ.
* CHESS\_BOARD\_WIDTH = 40;
* CHESS\_BOARD\_HEIGHT = 40;
* 3 thuộc tính xử lý thời gian người chơi chạy trên thanh ProgressBar.
* COOL\_DOWN\_STEP = 100;. Khoản thời gian chạy 1 lần.
* COOL\_DOWN\_TIME = 15000;. Thời gian tổng của thanh ProgressBar.
* COOL\_DOWN\_INTERVAL = 100;. Khoản cách của một step.
* Class Player
* Khởi tạo chứa ba thuộc tính name (tên người chơi), avatar (hình ảnh người chơi), symbol (kí tự người chơi đánh trên bàn cờ).
* string name;
* Image avatar;
* Image symbol;
* Class PlayInfo
* Khởi tạo chứa ba thuộc tính point (tọa độ người chơi), currentPlayer (để phân biệt được người đánh tiếp theo là ai), symbol (kí tự người chơi đánh trên bàn cờ).
* Point point;
* int currentPlayer;
* Image symbol;
* Class GameBoard
* Hàm DrawGameBoard()
* Thiết lập bàn cờ xóa sạch, bàn cờ trạng thái bằng true để người chơi đánh được và CurrentPlay = 0, gọi hàm ChangePlayer(). Thiết lập này giúp khi chơi một ván mới (NewGame) sẽ như ban đầu mở form.
* Tạo đối tượng PlayTimeLine, MatrixPositions, oldButton (dùng chứa vị trí ban đầu của một button).
* Tạo 2 vòng lặp for:
  + Duyệt từ i = 0 đến < CHESS\_BOARD\_WIDTH theo hàng dọc
  + Duyệt từ j = 0 đến < CHESS\_BOARD\_WIDTH theo hàng ngang
  + Ở hàng ngang khi một lần chạy khởi tạo đối tượng Button btn = new Button() chứa độ dài, độ rộng, vị trí (khởi tạo Point có tọa độ bằng oldButton.Location.X + oldButton.Width, oldButton.Location.Y. Sẽ tăng dần về bên phải, và thuộc tính Tag =i.ToString() để lưu lại danh sách button). Duyệt ra form và lưu vào mảng MatrixPositions[i]. Gọi hàm Btn\_Click nhận sự kiện khi click vào một button sẽ là hành động đánh của người chơi.
  + Khởi tạo lại oldButton có tọa độ là new Point(0, oldButton.Location.Y +CellHeight).
* Hàm Symbol()
* Thay đổi button backgroundimage đúng với trạng thái người chơi. Cập nhật lại trạng thái người chơi tiếp theo.
* btn.BackgroundImage = Players[CurrentPlayer].Symbol;
* CurrentPlayer = CurrentPlayer == 1 ? 0 : 1;
* Hàm ChangePlayer()
* In tên người chơi và ảnh thông qua trạng thái người chơi.
* PlayerName.Text = Players[CurrentPlayer].Name;
* Avatar.Image = Players[CurrentPlayer].Avatar;
* Hàm Btn\_Click()
* Kiểm tra ô đã được đánh chưa nếu rồi thì không được đánh. Thay đổi kí tự, tên, ảnh của người chơi và lưu vào trong Stack PlayTimeLine. Kiểm tra ô đã được đánh thì lưu vào hàm GetChessPoint(). Và xử lý kết thúc game.
* if (btn.BackgroundImage != null) return;
* Symbol(btn);
* PlayTimeLine.Push(new PlayInfo(GetChessPoint(btn),CurrentPlayer, btn.BackgroundImage));
* if (playerMarked != null) { playerMarked(this,new ButtonClickEvent(GetChessPoint(btn))); }
* if (isEndGame(btn)) EndGame();
* Hàm OtherPlayerClicked()
* Khởi tạo đối tượng btn để lấy các button được đánh của đối thủ trong mảng MatrixPositions. Kiểm tra ô đã được đánh chưa nếu rồi thì không được đánh. Thay đổi kí tự, tên, ảnh của người chơi và lưu vào trong Stack PlayTimeLine. Và xử lý kết thúc game.
* Button btn = MatrixPositions[point.Y][point.X];
* Symbol(btn);
* PlayTimeLine.Push(new PlayInfo(GetChessPoint(btn),CurrentPlayer, btn.BackgroundImage));
* if (isEndGame(btn)) EndGame();
* Hàm EndGame()
* Cho Timer dừng, in ra thông báo tên người chơi chiến thắng, và vẽ lại bàn cờ.
* TimerStop.Stop();
* MessageBox.Show(Players[CurrentPlayer == 1 ? 0 : 1].Name + " chiến thắng");
* DrawGameBoard();
* Hàm bool isEndGame()
* Chiến thắng nếu đúng 1 trong các trường hợp sau đây
* return isEndRow(btn) || isEndCol(btn) || isEndAuxiliaryDiagonal(btn) || isEndMainDiagonal(btn);
* Hàm Point GetChessPoint()
* Lấy tọa độ của các button và khởi tạo vào đối tượng point.
* int vertical = Convert.ToInt32(btn.Tag);
* int horizontal = MatrixPositions[vertical].IndexOf(btn);
* Point point = new Point(horizontal, vertical);
* Hàm bool isEndRow()
* Xử lý các button có cùng kí tự người chơi trên hàng ngang với kết quả >=5
* Point point = GetChessPoint(btn);
* Int countLeft=0, countRight=0;
* for (int i = point.X; i >= 0; i--)

{if(MatrixPositions[point.Y][i].BackgroundImage== btn.BackgroundImage)

{ countLeft++; }

else break;

}

* for (int i = point.X+1; i < Constant.CellWidth; i++)

{if(MatrixPositions[point.Y][i].BackgroundImage== btn.BackgroundImage)

{ countRight++; }

else break;

}

* return countLeft + countRight >= 5;
* Hàm bool isEndCol()
* Xử lý các button có cùng kí tự người chơi trên hàng dọc với kết quả >=5
* Point point = GetChessPoint(btn);
* Int countTop=0, countBottom=0;
* for (int i = point.Y; i >= 0; i--)

{if(MatrixPositions[i][point.X]BackgroundImage== btn.BackgroundImage)

{ countTop++; }

else break;

}

* for (int i = point.Y+1; i < Constant.CellHeight; i++)

{if(MatrixPositions[point.Y][i].BackgroundImage== btn.BackgroundImage)

{ countBottom++; }

else break;

}

* return countBottom + countTop >= 5;
* Hàm bool isEndMainDiagonal()
* Xử lý các button có cùng kí tự người chơi trên theo đường chéo chính với kết quả >=5
* Point point = GetChessPoint(btn);
* Int countTop=0, countBottom=0;
* for (int i =0; i <= point.X; i++)

{if(MatrixPositions[point.Y - i][point.X - i].BackgroundImage== btn.BackgroundImage)

{ countTop++; }

else break;

}

* for (int i = 1; i <= Constant.CellWidth - point.X; i++)

{if(MatrixPositions[point.Y + i][point.X + i].BackgroundImage== btn.BackgroundImage)

{ countBottom++; }

else break;

}

* return countTop + countBottom >= 5;
* Hàm bool isEndAuxiliaryDiagonal()
* Xử lý các button có cùng kí tự người chơi trên theo đường chéo phụ với kết quả >=5
* Point point = GetChessPoint(btn);
* for (int i = 0; i <= point.X; i++)

{if(MatrixPositions[point.Y - i][point.X + i].BackgroundImage== btn.BackgroundImage)

{ countTop++; }

else break;

}

* for (int i = 1; i <= Constant.CellWidth - point.X; i++)

{if(MatrixPositions[point.Y + i][point.X - i].BackgroundImage== btn.BackgroundImage)

{ countBottom++; }

else break;

}

* return countTop + countBottom >= 5;
* Hàm bool UndoAsStep()
* khởi tạo đối tượng PlayInfo oldPoint = PlayTimeLine.Pop();. Dùng Pop() là lấy hẳn một đối tượng ra khỏi Stack. Sau khi lấy ra cho button có BackgroundImage là null.
* PlayInfo oldPoint = PlayTimeLine.Pop();
* Button btn = MatrixPositions[oldPoint.Point.Y][oldPoint.Point.X];
* ChangePlayer();
* return true;
* Hàm bool Undo()
* Gọi lại hàm UndoAsStep 2 lần.
* return UndoAsStep() && UndoAsStep();
* Class SocketManager
* Hàm ConnectServer() sử dụng lớp Socket client
* 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 true; }

* Hàm CreateServer() sử dụng lớp Socket server.
* 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();

* Hàm byte[] SerializeData()
* MemoryStream ms = new MemoryStream();
* BinaryFormatter bf1 = new BinaryFormatter();
* bf1.Serialize(ms, o);
* return ms.ToArray();
* Hàm DeserializeData()
* MemoryStream ms = new MemoryStream(theByteArray);
* BinaryFormatter bf1 = new BinaryFormatter();
* ms.Position = 0;
* return bf1.Deserialize(ms);
* Hàm SendData()
* return target.Send(data) == 1 ? true : false;
* Hàm ReceiveData()
* return target.Receive(data) == 1 ? true : false;
* Hàm Send()
* byte[] sendData = SerializeData(data);
* return SendData(client, sendData);
* Hàm Receive()
* byte[] receiveData = new byte[BUFFER];
* bool isOk = ReceiveData(client, receiveData);
* return DeserializeData(receiveData);
* Hàm GetLocalIPv4()
* 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;

* Class SocketData
* int command;
* Point point;
* string message;
* Hàm liệt kê enum SocketCommand
* SEND\_POINT,

NOTIFY,

NEW\_GAME,

UNDO,

END\_GAME,

TIME\_OUT,

QUIT

* Hàm form\_shown()
* txt\_IP.Text = socketManager.GetLocalIPv4(NetworkInterfaceType.Wireless80211);
* if (string.IsNullOrEmpty(txt\_IP.Text))

{

txt\_IP.Text = socketManager.GetLocalIPv4(NetworkInterfaceType.Ethernet);

}

* Hàm btn\_Lan\_Click()
* socketManager.IP = txt\_IP.Text;
* if (socketManager.ConnectServer() == false)

{

socketManager.isServer = true;

pn\_GameBoard.Enabled = true;

socketManager.CreateServer();

btn\_LAN.Enabled = false;

}

else

{

socketManager.isServer = false;

pn\_GameBoard.Enabled = false;

MessageBox.Show("Kết nối thành công!");

btn\_LAN.Enabled = false;

Listen();

}

* Hàm Listen()
* try { Thread listernThread = new Thread(() =>

{

While(true){

try{

SocketData data = (SocketData) socketManager. eceive();

ProcessData(data);

break;

}

Catch{}

Thread.Sleep(10);

}

};

listernThread.IsBackground = true;

listernThread.Start();

}catch{}

* Hàm ProcessData()

|  |
| --- |
| switch (data.Command)  {  case (int)SocketCommand.NOTIFY:  MessageBox.Show(data.Message);  break;  case (int)SocketCommand.NEW\_GAME:  this.Invoke((MethodInvoker)(() =>  {  NewGame();  pn\_GameBoard.Enabled = false;  }));  break;  case (int)SocketCommand.SEND\_POINT:  this.Invoke((MethodInvoker)(() => {  prcbCoolDownZero();  pn\_GameBoard.Enabled = true;  btn\_Undo.Enabled = true;  tmCoolDown.Start();  chessBoardManager.OtherPlayerClicked(data.Point);  }));  break;  case (int)SocketCommand.UNDO:  chessBoardManager.Undo();  prcbCoolDownZero();  break;  case (int)SocketCommand.TIME\_OUT:  string name = chessBoardManager.Players[chessBoardManager.CurrentPlayer == 1 ? 0 : 1].Name;  tmCoolDown.Stop();  MessageBox.Show(name + " Thắng cuộc");  EndGame();    break;  case (int)SocketCommand.QUIT:  tmCoolDown.Stop();  MessageBox.Show("Người chơi đã thoát");  break;  default:  break;  }  Listen(); |

* Hàm ChessBoard\_PlayerMarked()

|  |
| --- |
| tmCoolDown.Start();  pn\_GameBoard.Enabled = false;  btn\_Undo.Enabled = false;  prcbCoolDownZero();  socketManager.Send(new SocketData((int)SocketCommand.SEND\_POINT, "", e.ClickedPoint));  Listen(); |

* Hàm tmCoolDown\_Tick()

|  |
| --- |
| prcbCoolDown.PerformStep();  if (prcbCoolDown.Value >= prcbCoolDown.Maximum)  {  EndGame();  socketManager.Send(new SocketData((int)SocketCommand.TIME\_OUT, "", new Point()));  } |

* Hàm Form\_KeyDown()

|  |
| --- |
| * if (e.Control == true && e.KeyCode == Keys.Q)   { Quit();}   * if (e.Control == true && e.KeyCode == Keys.N)   { NewGame(); }   * if (e.Control == true && e.KeyCode == Keys.Z)   { Undo(); } |

* Hàm EndGame()
  + tmCoolDown.Stop();
  + pn\_GameBoard.Enabled = false;
* Hàm NewGame()
* prcbCoolDownZero();
* tmCoolDown.Stop();
* chessBoardManager.DrawGameBoard();
* Hàm Quit()
* if (MessageBox.Show("Bạn có chắc muốn thoát chương trình ?", "Thông báo", MessageBoxButtons.OKCancel) == System.Windows.Forms.DialogResult.OK)

Application.Exit();

* Hàm Form\_Load
  + this.KeyPreview = true;
* Hàm Undo()
  + prcbCoolDownZero();
  + chessBoardManager.Undo();
  + socketManager.Send(new SocketData((int)SocketCommand.UNDO, "", new Point()));
* Hàm Form\_Closing
  + if (MessageBox.Show("Bạn có chắc muốn thoát chương trình ?", "Thông báo", MessageBoxButtons.OKCancel) != System.Windows.Forms.DialogResult.OK)

e.Cancel = true;

else{

Try{

socketManager.Send(new SocketData((int)SocketCommand.QUIT, "", new Point()));}

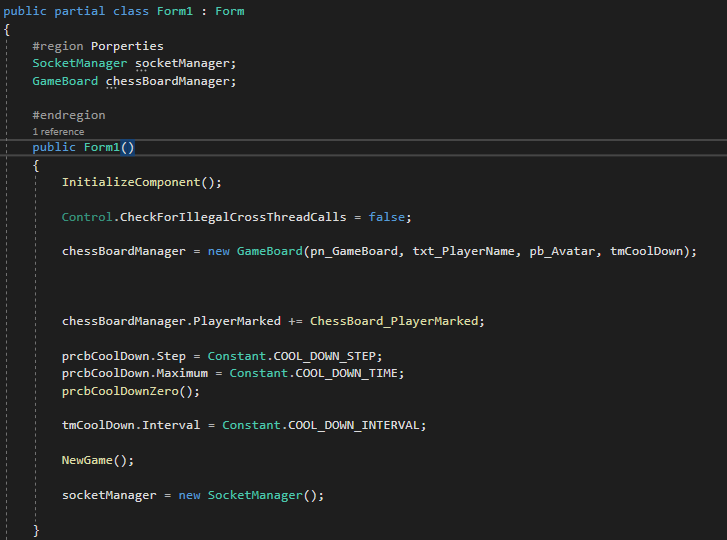
catch{}

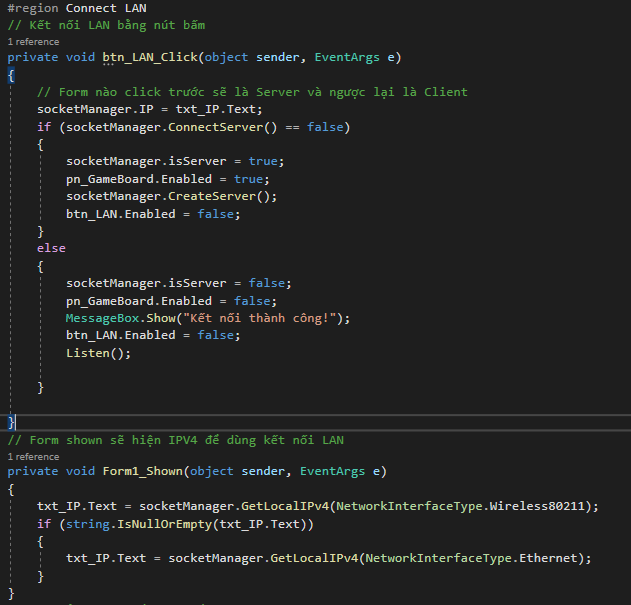
}

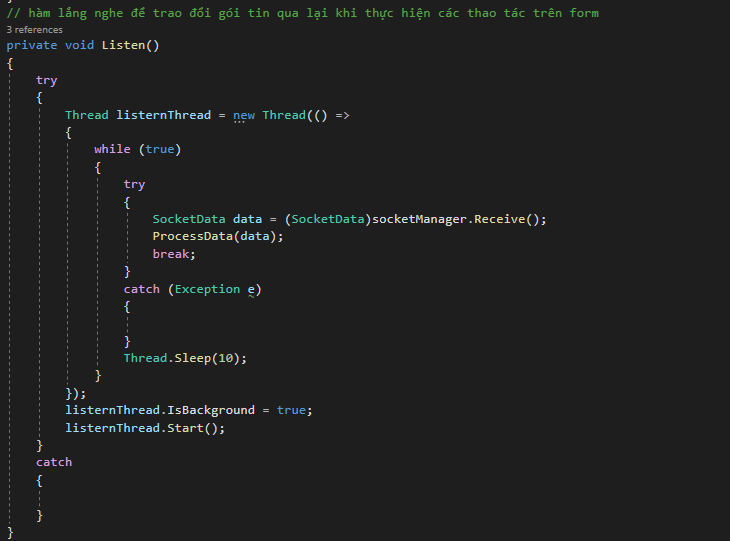
* Hàm prcbCoolDownZero
  + prcbCoolDown.Value = 0;
* Hàm btn\_Undo\_Click()
  + Undo();
* Hàm btnNewgame\_Click()
  + NewGame();
  + socketManager.Send(new SocketData((int)SocketCommand.NEW\_GAME, "", new Point()));
  + pn\_GameBoard.Enabled = true;
* Hàm btnQuit\_Click()
  + Quit();

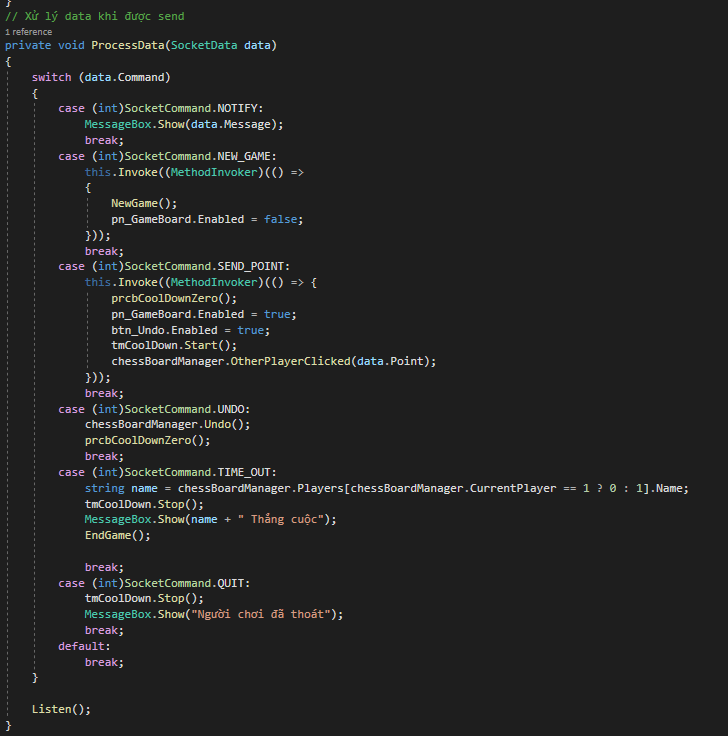
# Cài đặt bài toán

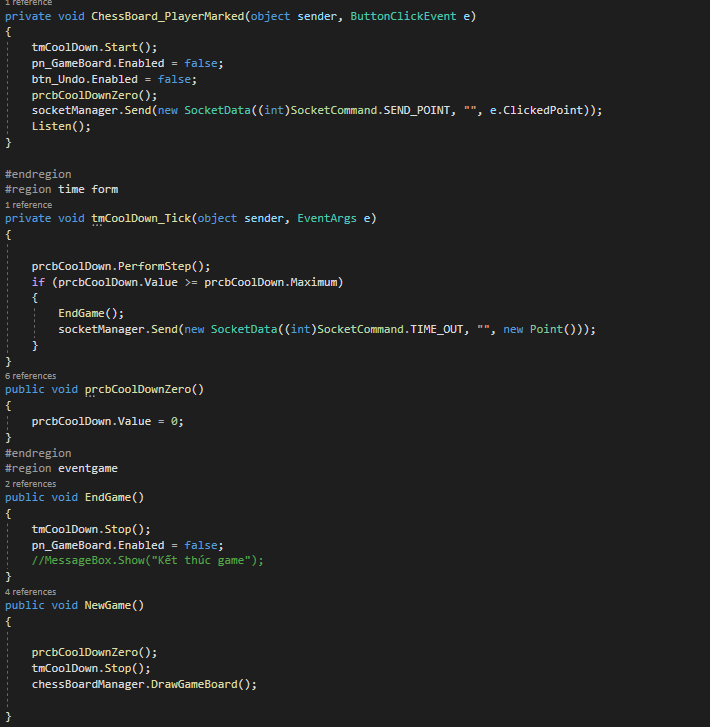
* **Class Form**

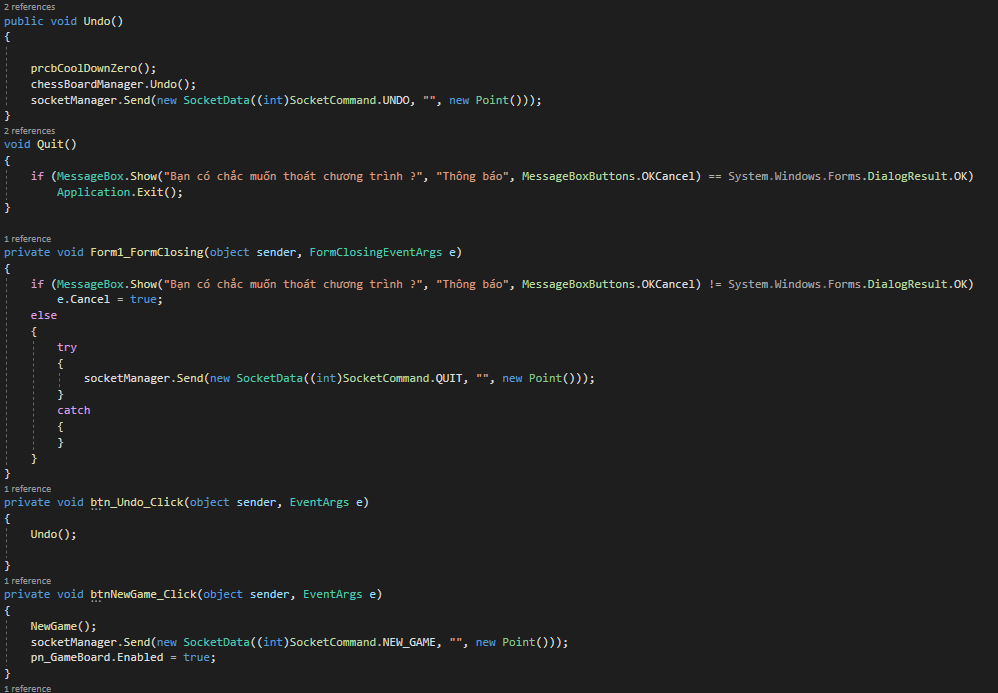
****

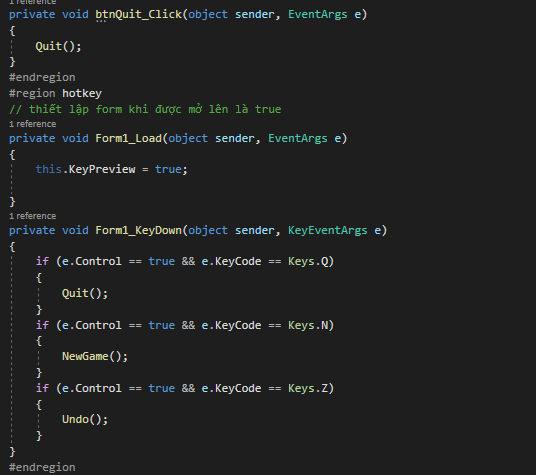
****

****

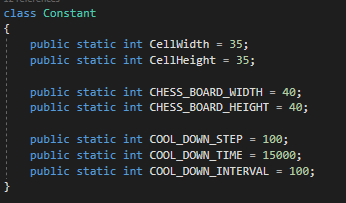
****

****

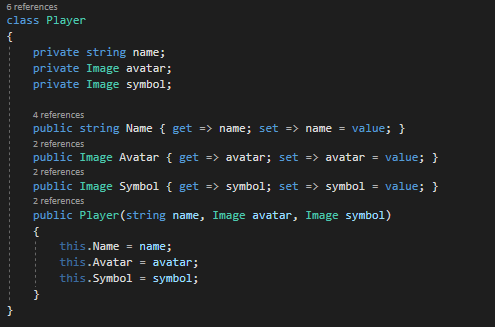
****

****

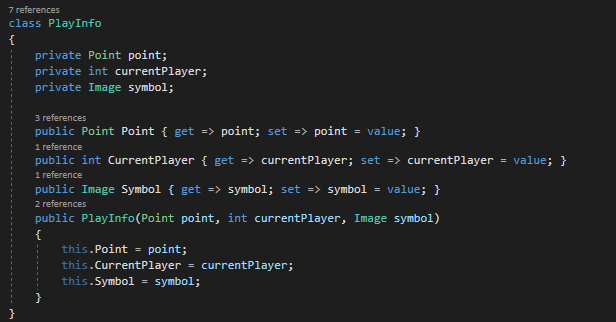
* **Class Contant**

****

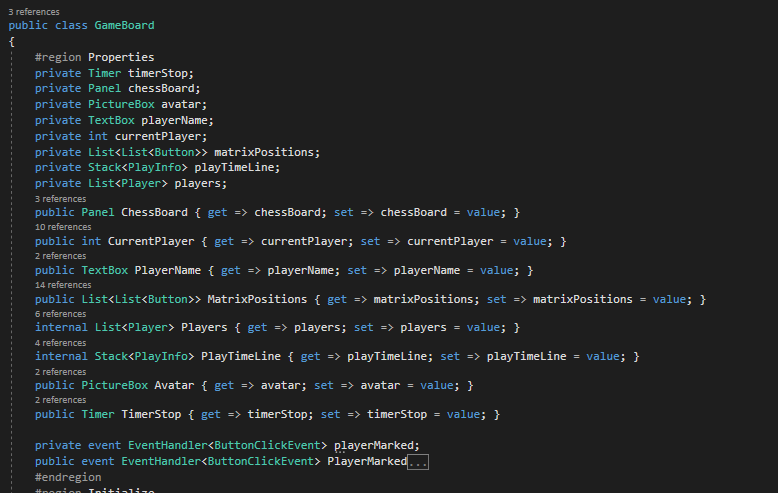
* **Class Player**

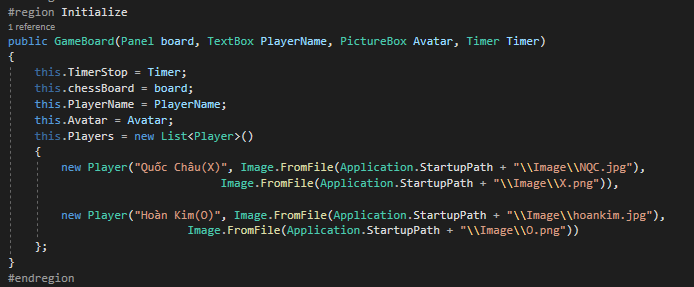
****

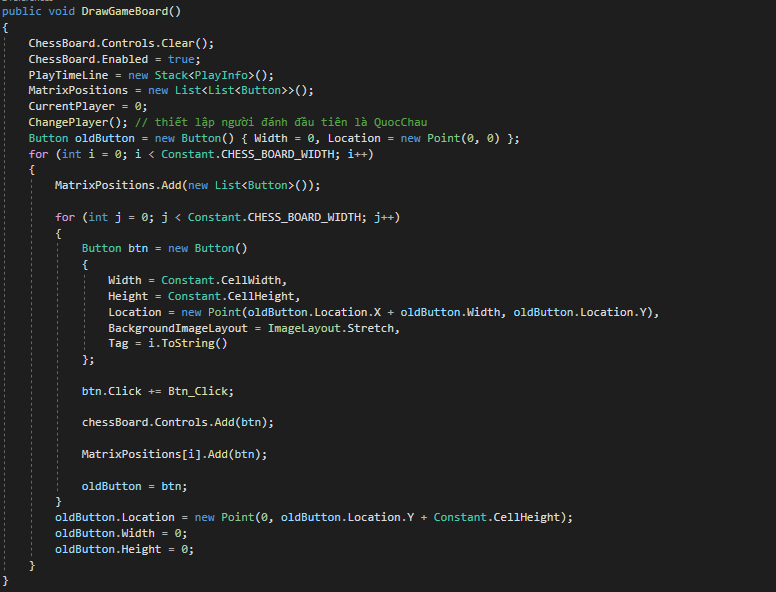
* **Class PlayInfo**

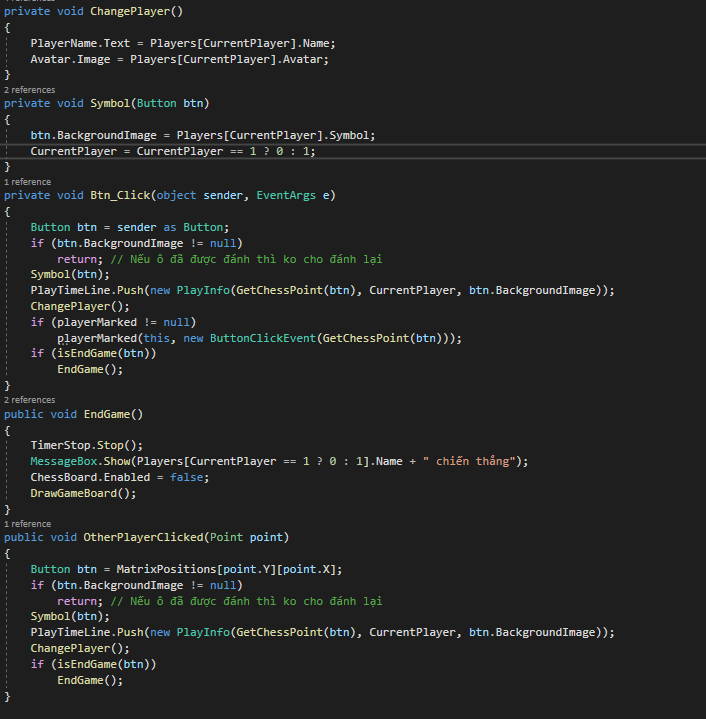
****

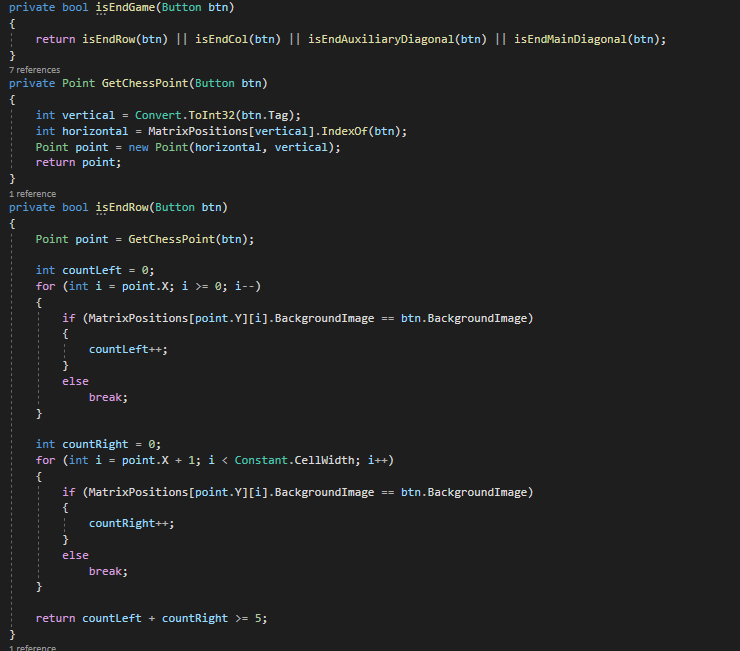
* **Class GameBoard**

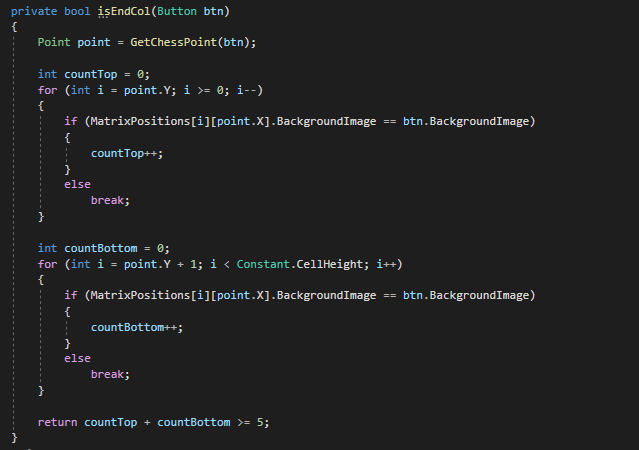
****

****

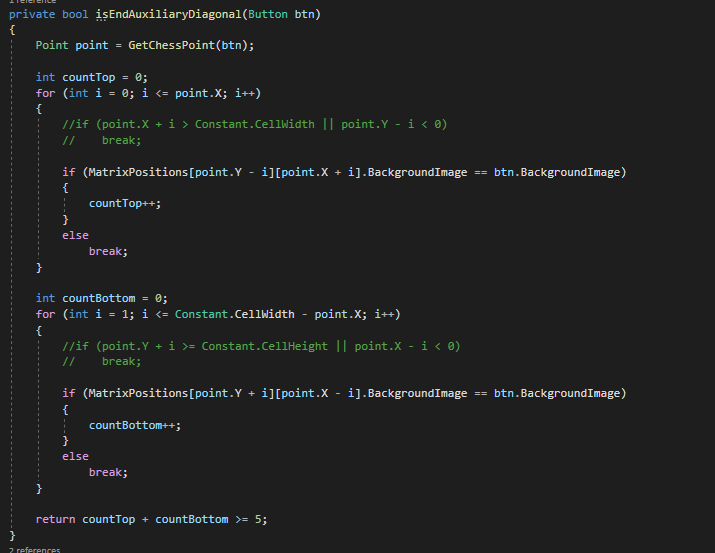
**s**

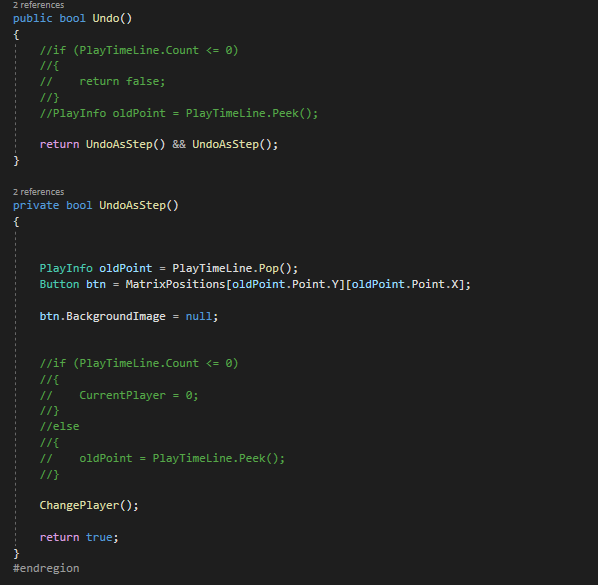
****

****

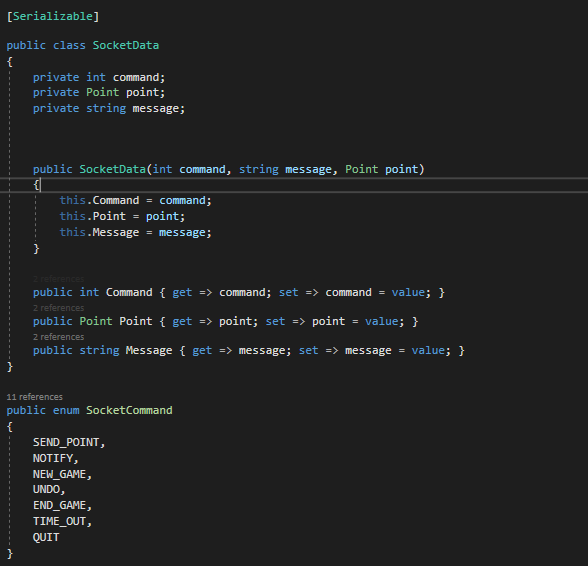
****

****

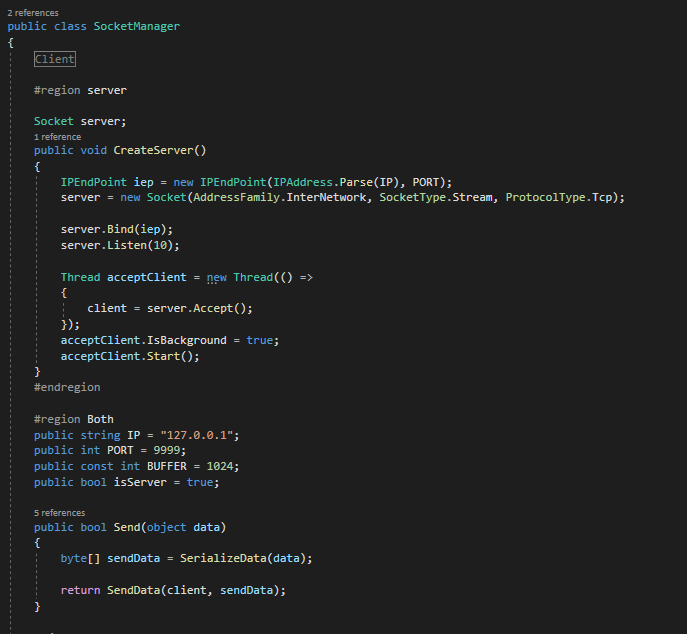
****

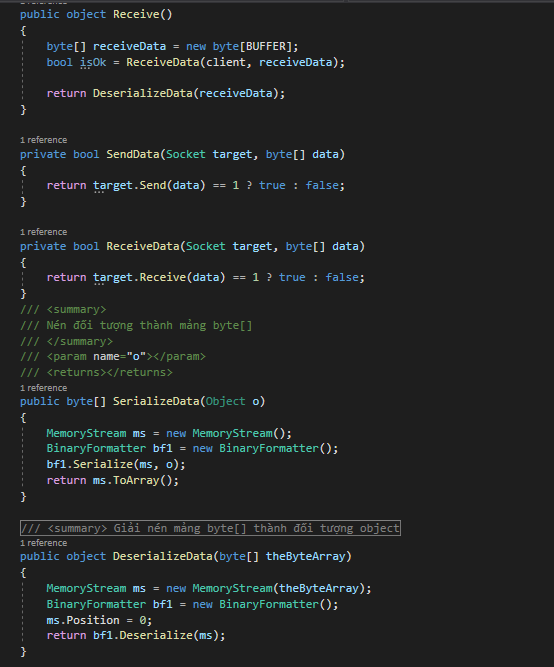
****

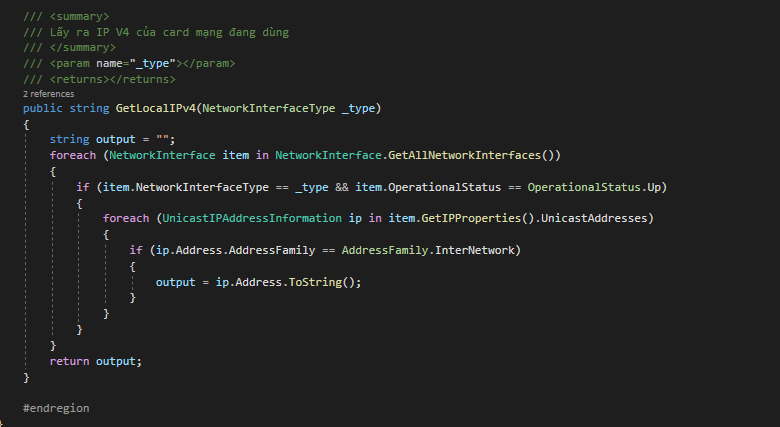
* **Class SocketData**

****

* **Class SocketManager**

****

****

****

# Kết quả cài đặt