PROGRAMMING

TRINH THI DIEU HUYEN
GDD18606

ASSIGNMENT 2 FRONT SHEET

Qualification	**BTEC Level 5 HND Diploma in Computing**		
Unit number and title	Unit 1: Programming		
Submission date	28/09/2019	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Student Name	Trinh Thi Dieu Huyen	**Student ID**	GDD18606
Class	GCD0826	**Assessor name**	Tran Trong Minh

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.

Student's signature	

Grading grid

P2	P3	P4	P5	M2	M3	M4	D2	D3	D4

Summative Feedback:	**Resubmission Feedback:**

Grade:	**Assessor Signature:**	**Date:**
Lecturer Signature:		

Table of content

A. Explain the characteristics of procedural, object-oriented and event-driven programming, conduct an analysis of a suitable Integrated Development Environment (IDE)....5

I. Give explanations of what procedural, object-oriented and event-driven paradigms are; their characteristics and the relationship between them....5
II. Analyse the common features that a developer has access to in an IDE....6
III. Critically evaluate the source code of an application which implements the programming paradigms, in terms of the code structure and characteristics....8

B. Implement basic algorithms in code using an IDE....9

I. Write a program that implements an algorithm using an IDE and Use the IDE to manage the development process of the program....9
II. Evaluate the use of an IDE for development of applications contrasted with not using an IDE....45

C. Determine the debugging process and explain the importance of a coding standard....46

I. Explain the debugging process and explain the debugging facilities available in the IDE....47
II. Outline the coding standard you have used in your code....48
III. Evaluate how the debugging process can be used to help develop more secure, robust applications....49
IV. Critically evaluate why a coding standard is necessary in a team as well as for the individual....49

D. Reference....50

A. Explain the characteristics of procedural, object-oriented and event-driven programming, conduct an analysis of a suitable Integrated Development Environment (IDE).

I. Give explanations of what procedural, object-oriented and event-driven paradigms are; their characteristics and the relationship between them.

- In procedural programming paradigm, one defines a program and its subprograms as a series of steps.

- In event-driven programming paradigm, one defines what will occur when a user executes an event.

- Object-oriented programming paradigm is a way of organizing code around a collection of objects based on the principles of encapsulation, inheritance, substitution, programming to interfaces, and so on.

- Procedural, event-driven and object-oriented paradigms are used to classify programming languages.
- A procedural programming paradigm is represented as a series of steps in which a computer programmer writes a code to perform a specific task. The first line of code is executed first, then the second line of code, then the third, up to last line of code. Therefore, procedural programming is a top-down approach to executing lines of codes. The most popular languages that use procedural paradigm are BASIC, COBOL, PASCAL and FORTRAN. In these languages, a program comprises of steps or procedures that operate on data. Procedural programming has many characteristics like local variables, global variables, pre-defined functions, modularity, and parameter passing in function and procedures.

- While the procedural paradigm uses steps, the event-driven paradigm approach uses events from the user to execute functions on the data. These events could be a simple mouse click or a combination of key-presses from the user (like in GUI based programs) or could also be a message from other programs. Programming languages constantly listen or sense for an event, which then calls the respective function when the event is detected. Since most of the GUI applications depend on user actions, these GUI programming languages will have an event-driven paradigm based programming as an essential part. The characteristics of event-driven programming are that these are service oriented, use event handlers and triggers, have user-interfaces and are time driven. Most popular languages specially for GUI or visual applications that use event-driven paradigm are visual C++, Java, Visual Basic.

- The object-oriented paradigm is different from both the steps based paradigm of procedural and the trigger/event based paradigm of event-driven, in the sense that the object-oriented paradigm is based on objects and classes and their re-use to perform a specific code-function. So, unlike the procedural paradigm, the object-oriented paradigm uses bottom-up approach. In the simplest form, these objects contain data in the form of fields. Programs are divided into smaller units known as objects belonging to a class. These objects interact with each other through functions. Most popular languages that use object-oriented paradigm are C++, Java, Objective-C, Python. The characteristics of languages using the object-oriented paradigms are the use of objects, classes, encapsulation, inheritance, data abstraction, dynamic binding, message passing and polymorphism.

II. Analyse the common features that a developer has access to in an IDE.

❖ **<u>Common features of integrated development environments:</u>**

- An IDE typically contains a code editor, a compiler or interpreter, and a debugger, accessed through a single graphical user interface (GUI). The user writes and edits source code in the code editor. The compiler translates the source code into a readable language that is executable for a computer. And the debugger tests the software to solve any issues or bugs.

- An IDE can also contain features such as programmable editors, object and data modeling, unit testing, a source code library and build automation tools.

- An IDE's toolbar looks much like a word processor's toolbar. The toolbar facilitates color-based organization, source-code formatting, error diagnostics and reporting, and intelligent code completion. Through an IDE's interface, a developer or team of developers can compile and execute code incrementally and manage changes to source code in a uniform manner. IDEs are typically designed to integrate with third-party version control libraries, such as GitHub and Apache's Subversion.

- An IDE can support model-driven development (MDD). A developer working with an IDE starts with a model, which the IDE translates into suitable code. The IDE then debugs and tests the model-driven code, with a high level of automation. Once the build is successful and properly tested, it can be deployed for further testing through the IDE or other tools outside of the IDE.

❖ **Benefits of using IDEs:**

- An IDE can improve the productivity of software developers thanks to fast setup and standardization across tools.

- Without an IDE, developers spend time deciding what tools to use for various tasks, configuring the tools and learning how to use them. Many or even all of the necessary dev-test tools are included in one integrated development environment.

- IDEs are also designed with all their tools under one user interface. An IDE can standardize the development process by organizing the necessary features for software development in the UI.

❖ **Types of IDEs and available tools:**

- Developers must match the IDE they use with the type of application they want to produce. For example, if a developer wants to create an application on iOS, then they need an IDE that supports Apple's Swift programming language. Types of IDEs range from web-based and cloud-based to mobile, language-specific or multi-language.

- Web-based IDEs suit web-based application development in HTML, JavaScript or similar programming languages. Microsoft's Visual Studio Code is an example of a web-based IDE with features such as a code editor, syntax highlighting, code completion and debugging.

- Increasingly, IDEs are offered on a platform as a service (PaaS) delivery model. The benefits of these cloud-based IDEs include accessibility to software development tools from anywhere in the world, from any compatible device; minimal to nonexistent download and installation requirements; and ease of collaboration among geographically dispersed developers. Cloud9 is an IDE from AWS that supports up to 40 languages including C, C++, Python, Ruby and JavaScript. Cloud9 gives users code completion, an image editor and a debugger, as well as other features such as support for deployment to Microsoft Azure and Heroku (which is a cloud-based PaaS IDE).

- An IDE for mobile development normally works with code that runs on iOS or Android devices. Xamarin is an example of a cross-platform mobile IDE, which means it can create code for multiple mobile platform types. For example, a developer can write a feature in C and Xamarin translates it into Swift for iOS and Java for Android. Additionally, Xamarin offers UI tests and it can distribute beta tests to users.

- IDEs such as C-Free -- which supports a code editor, debugger and an environment to run C and C++ code -- are language specific. Other IDEs support multiple languages, such as previously mentioned Cloud9 and Visual Studio Code. More popular IDE tools include NetBeans, Eclipse and IntelliJ IDEA.

- Disambiguation: The abbreviation IDE is also used to refer to integrated drive electronics.

III. Critically evaluate the source code of an application which implements the programming paradigms, in terms of the code structure and characteristics.

- A "programming paradigm" is a primary style of computer programming. Different methodologies are appropriate for solving particular problems or developing applications in different domains. In computer programming, source code is a collection of code, written in a human-readable programming language.

- Computer programming languages permit your to give instructions to a computer in a language the computer understands. At a basic level, computers only understand binary language (1's and 0's), which makes no sense to humans. There are many programming languages in the computer world.

- In general, a computer program consists of the following components:

  - Program Structure: The general type of a program, there is always a structure or format in which the programmer develop programs. A well-organized program uses suitable information structures and formats.

- Variable Declaration: A variable is used to store values at run time or time of execution. Some programming languages give flexibility to the programmer to avoid variable declaration, but a good programming language must have a proper variable declaration method.
- Looping structures: Looping in computer programming is used to repeat statements, under a given condition. There are many types of loops with minor difference in different computer programming languages. Loops will keep executing until the exit condition is met.
- Control structures: A control structure is a slab in computer programming that examines variables and chooses the execution direction in which to go, based on given parameters or conditions.
- Sentence structure: In computer programming, when a programmer writes a program they use programming notation in an expository. Thus, it is important that the notation has a sentence structure that can be easily expressed.

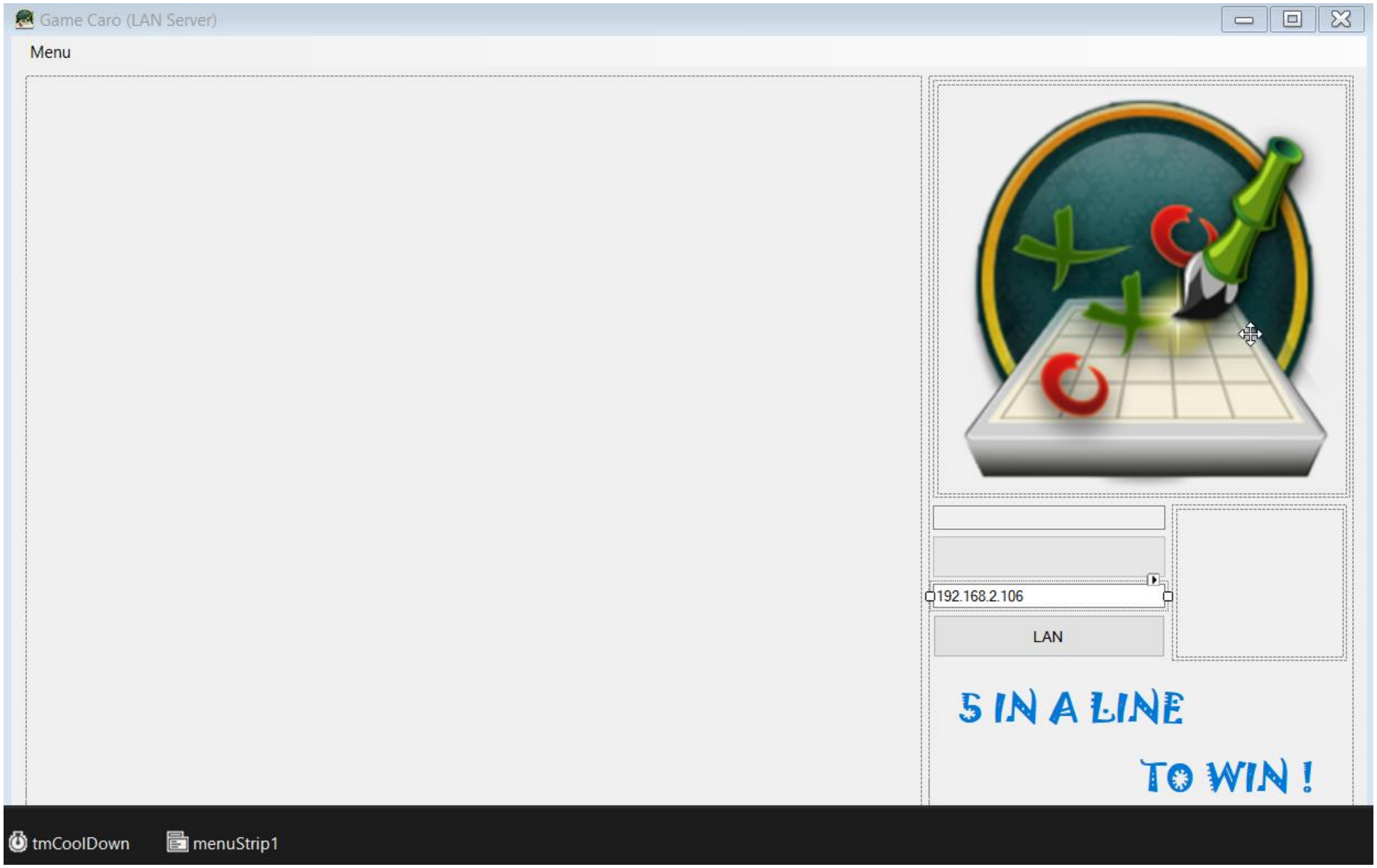
B. Implement basic algorithms in code using an IDE.

I. Write a program that implements an algorithm using an IDE and Use the IDE to manage the development process of the program.

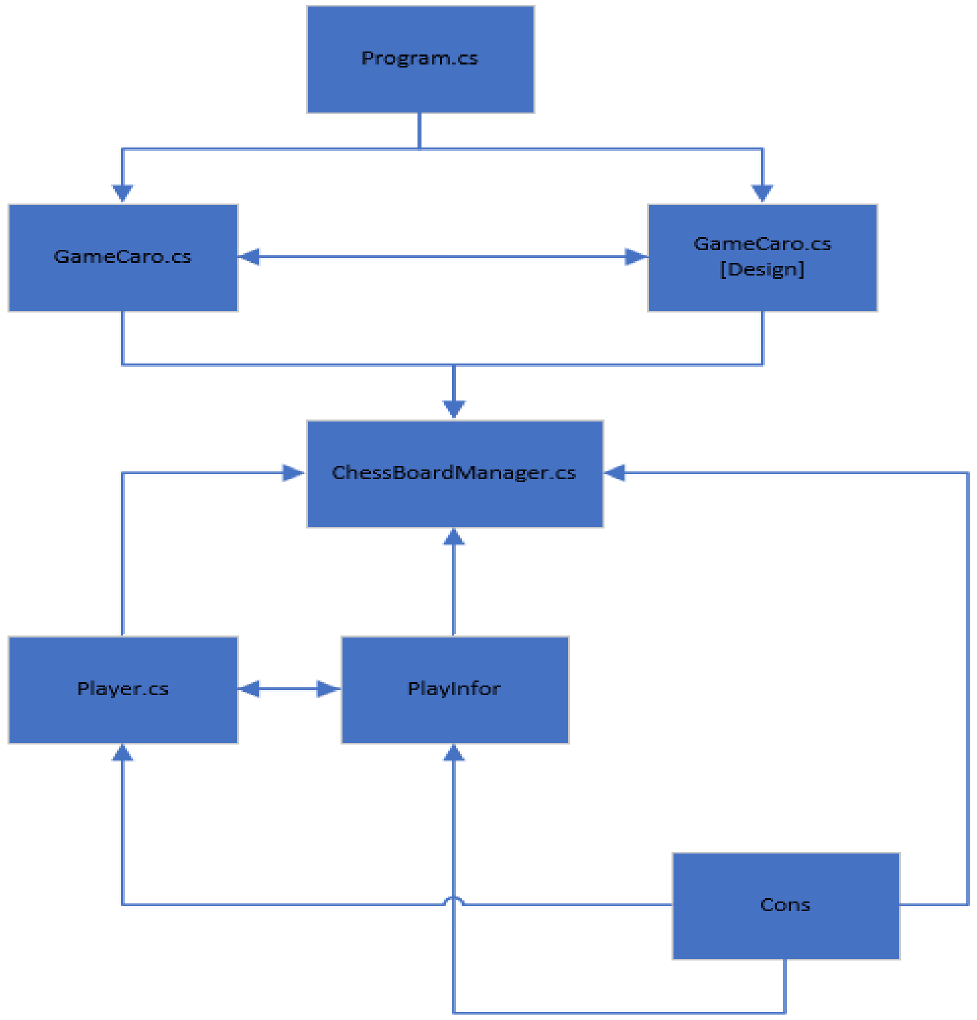
<u>Project name:</u> Create Caro chess software one person.

❖ **<u>Use:</u>**

- Panel & picturebox: contains chess board (buttons) and images.
- Menuscrip: create selective forces like New game, Undo, Quit.
- Button: contains information to fill and create a chessboard.
- Textbox: contains information to fill.
- ProgressBar & timer: Run time thinking.



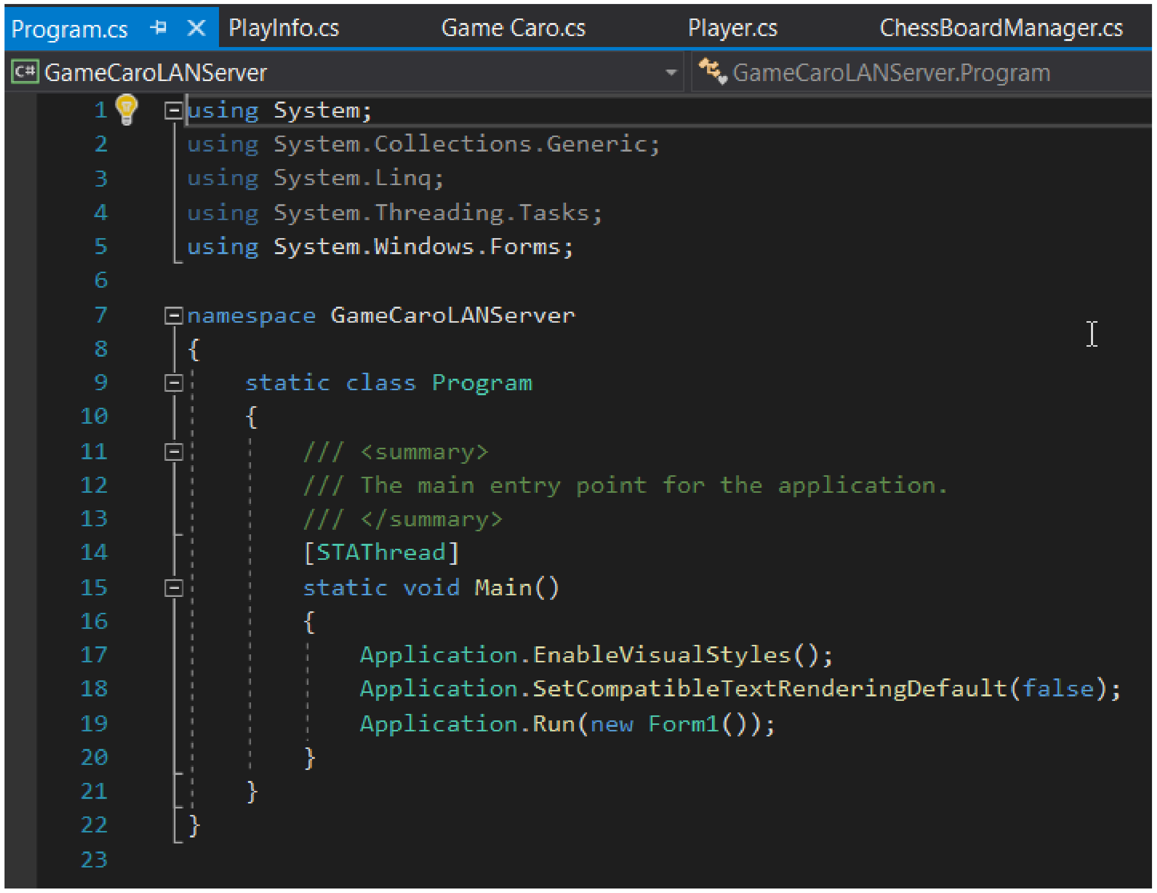
Display Design.



Activity diagram.

❖ **CODE:**

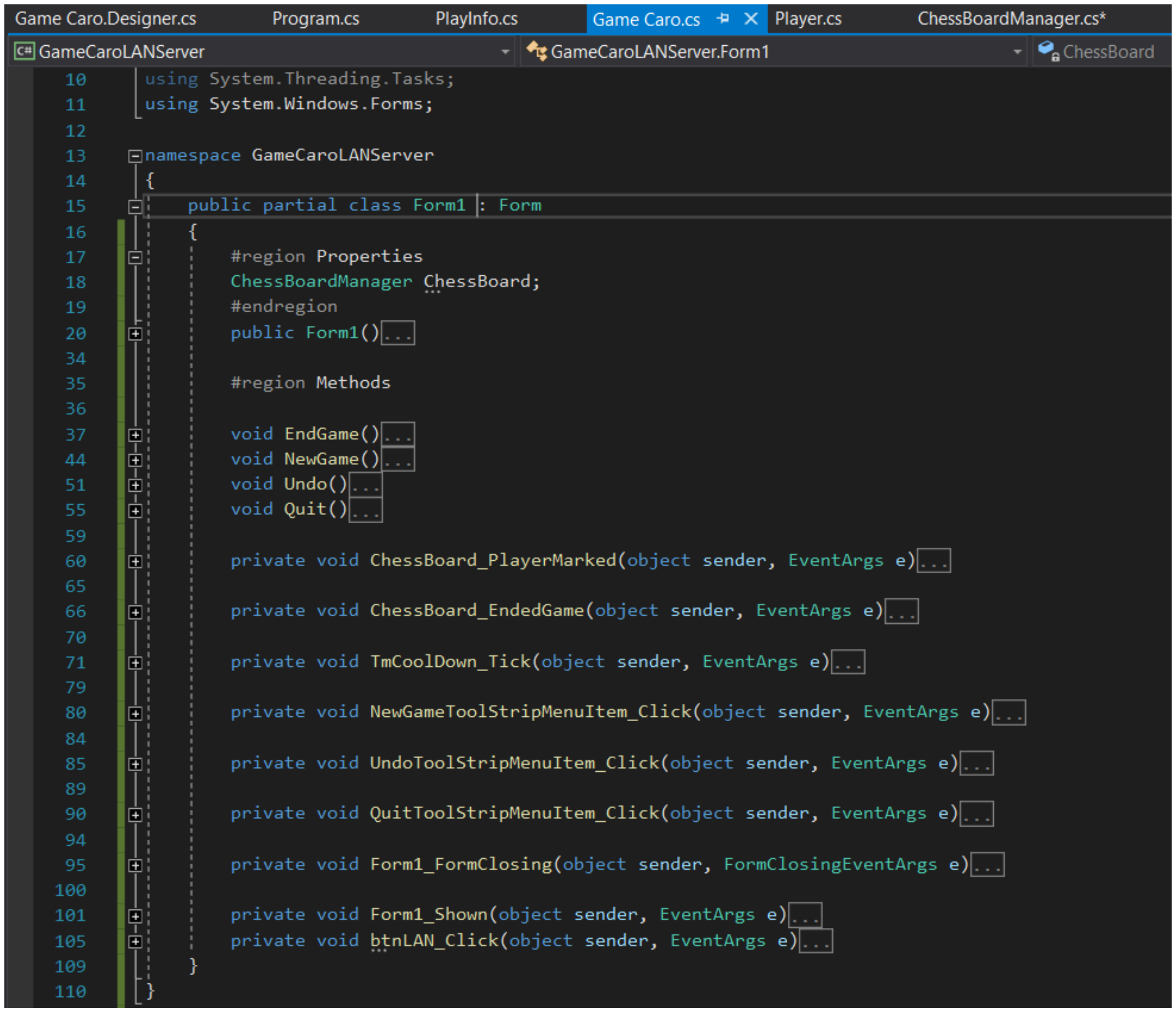
a. Program.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GameCaroLANServer
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Image Program.cs

b. GameCaro.



Game Caro.Designer.cs | Program.cs | PlayInfo.cs | Game Caro.cs | Player.cs | ChessBoardManager.cs*

GameCaroLANServer | GameCaroLANServer.Form1 | ChessBoard

```
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GameCaroLANServer
{
    public partial class Form1 : Form
    {
        #region Properties
        ChessBoardManager ChessBoard;
        #endregion
        public Form1()...

        #region Methods

        void EndGame()...
        void NewGame()...
        void Undo()...
        void Quit()...

        private void ChessBoard_PlayerMarked(object sender, EventArgs e)...

        private void ChessBoard_EndedGame(object sender, EventArgs e)...

        private void TmCoolDown_Tick(object sender, EventArgs e)...

        private void NewGameToolStripMenuItem_Click(object sender, EventArgs e)...

        private void UndoToolStripMenuItem_Click(object sender, EventArgs e)...

        private void QuitToolStripMenuItem_Click(object sender, EventArgs e)...

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)...

        private void Form1_Shown(object sender, EventArgs e)...
        private void btnLAN_Click(object sender, EventArgs e)...
    }
}
```

GameCaro.cs

- Code detail:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.NetworkInformation;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GameCaroLANServer
{
    public partial class Form1 : Form
    {
        #region Properties
        ChessBoardManager ChessBoard;
        #endregion
        public Form1()
        {
            InitializeComponent();

            ChessBoard = new ChessBoardManager(pnlChessBoard, txbPlayerName, pctbMark);
            ChessBoard.EndedGame += ChessBoard_EndedGame;
            ChessBoard.PlayerMarked += ChessBoard_PlayerMarked;
            prcbCoolDown.Step = Cons.COOL_DOWN_STEP;
            prcbCoolDown.Maximum = Cons.COOL_DOWN_TIME;
            prcbCoolDown.Value = 0;
            tmCoolDown.Interval = Cons.COOL_DOWN_INTERVAL;
            NewGame();

        }
```

```
#region Methods

void EndGame()
{
    tmCoolDown.Stop();
    pnlChessBoard.Enabled = false;
    undoToolStripMenuItem.Enabled = false;
    MessageBox.Show("Finish Game!");
}
void NewGame()
{
    prcbCoolDown.Value = 0;
    tmCoolDown.Stop();
    undoToolStripMenuItem.Enabled = true;
    ChessBoard.DrawChessBroard();
}
void Undo()
{
    ChessBoard.Undo();
}
void Quit()
{
   Application.Exit();
}

private void ChessBoard_PlayerMarked(object sender, EventArgs e)
{
    tmCoolDown.Start();
    prcbCoolDown.Value = 0;
}

private void ChessBoard_EndedGame(object sender, EventArgs e)
{
    EndGame();
}
```

```
        private void TmCoolDown_Tick(object sender, EventArgs e)
        {
            prcbCoolDown.PerformStep();
            if (prcbCoolDown.Value >= prcbCoolDown.Maximum)
            {
                EndGame();
            }
        }

        private void NewGameToolStripMenuItem_Click(object sender, EventArgs e)
        {
            NewGame();
        }
        private void UndoToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Undo();
        }
        private void QuitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Quit();
        }
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (MessageBox.Show("Are you sure you want to quit the game?", "Notification", MessageBoxButtons.OKCancel) !=
System.Windows.Forms.DialogResult.OK)
            e.Cancel = true;
        }
        private void Form1_Shown(object sender, EventArgs e)
        {

        }
        private void btnLAN_Click(object sender, EventArgs e)
        {
            #endregion
        }
    }
}
```

c. GameCaro Design.

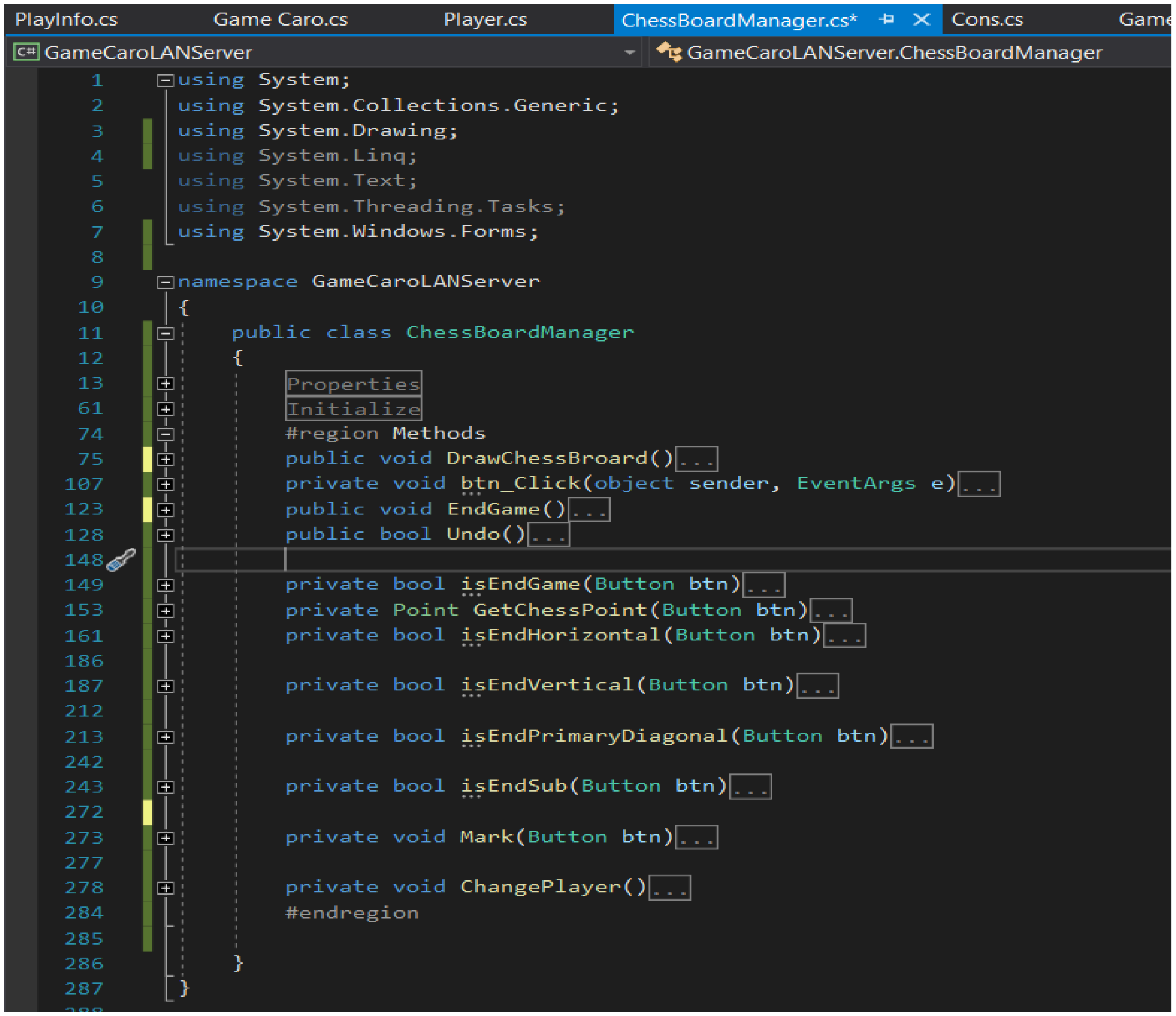
```
namespace GameCaroLANServer
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
            this.pnlChessBoard = new System.Windows.Forms.Panel();
            this.panel2 = new System.Windows.Forms.Panel();
            this.label2 = new System.Windows.Forms.Label();
            this.label1 = new System.Windows.Forms.Label();
            this.btnLAN = new System.Windows.Forms.Button();
            this.txbIP = new System.Windows.Forms.TextBox();
            this.prcbCoolDown = new System.Windows.Forms.ProgressBar();
            this.txbPlayerName = new System.Windows.Forms.TextBox();
            this.panel4 = new System.Windows.Forms.Panel();
            this.panel3 = new System.Windows.Forms.Panel();
            this.pctbMark = new System.Windows.Forms.PictureBox();
            this.pictureBox1 = new System.Windows.Forms.PictureBox();
            this.tmCoolDown = new System.Windows.Forms.Timer(this.components);
            this.menuStrip1 = new System.Windows.Forms.MenuStrip();
            this.menuToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.newGameToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.undoToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.quitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.panel2.SuspendLayout();
            this.panel4.SuspendLayout();
            this.panel3.SuspendLayout();
            ((System.ComponentModel.ISupportInitialize)(this.pctbMark)).BeginInit();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
            this.menuStrip1.SuspendLayout();
            this.SuspendLayout();
            //
            // pnlChessBoard
            //
            this.pnlChessBoard.Location = new System.Drawing.Point(13, 36);
            this.pnlChessBoard.Name = "pnlChessBoard";
            this.pnlChessBoard.Size = new System.Drawing.Size(814, 676);
            this.pnlChessBoard.TabIndex = 0;
            // 
            // panel2
            // 
            this.panel2.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms.AnchorStyles.Top | 
System.Windows.Forms.AnchorStyles.Right)));
            this.panel2.Controls.Add(this.label2);
            this.panel2.Controls.Add(this.label1);
            this.panel2.Controls.Add(this.btnLAN);
            this.panel2.Controls.Add(this.txbIP);
            this.panel2.Controls.Add(this.prcbCoolDown);
            this.panel2.Controls.Add(this.txbPlayerName);
            this.panel2.Controls.Add(this.panel4);
            this.panel2.Controls.Add(this.panel3);
            this.panel2.Location = new System.Drawing.Point(833, 36);
            this.panel2.Name = "panel2";
            this.panel2.Size = new System.Drawing.Size(386, 676);
            this.panel2.TabIndex = 1;
            // 
            // label2
            // 
            this.label2.AutoSize = true;
            this.label2.Font = new System.Drawing.Font("Jokerman", 19.8F, System.Drawing.FontStyle.Bold, 
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label2.ForeColor = System.Drawing.SystemColors.MenuHighlight;
            this.label2.Location = new System.Drawing.Point(183, 613);
            this.label2.Name = "label2";
            this.label2.Size = new System.Drawing.Size(181, 49);
            this.label2.TabIndex = 7;
            this.label2.Text = "TO WIN !";
            //
            // label1
            //
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Jokerman", 19.8F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label1.ForeColor = System.Drawing.SystemColors.MenuHighlight;
            this.label1.Location = new System.Drawing.Point(18, 550);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(227, 49);
            this.label1.TabIndex = 6;
            this.label1.Text = "5 IN A LINE";
            //
            // btnLAN
            //
            this.btnLAN.Location = new System.Drawing.Point(4, 489);
            this.btnLAN.Name = "btnLAN";
            this.btnLAN.Size = new System.Drawing.Size(211, 39);
            this.btnLAN.TabIndex = 5;
            this.btnLAN.Text = "LAN";
            this.btnLAN.UseVisualStyleBackColor = true;
            //
            // txbIP
            //
            this.txbIP.Location = new System.Drawing.Point(4, 461);
            this.txbIP.Name = "txbIP";
            this.txbIP.Size = new System.Drawing.Size(211, 22);
            this.txbIP.TabIndex = 4;
            this.txbIP.Text = "192.168.2.106";
            //
            // prcbCoolDown
            //
```

```
this.prcbCoolDown.Location = new System.Drawing.Point(4, 418);
this.prcbCoolDown.Name = "prcbCoolDown";
this.prcbCoolDown.Size = new System.Drawing.Size(211, 37);
this.prcbCoolDown.TabIndex = 3;
//
// txbPlayerName
//
this.txbPlayerName.Location = new System.Drawing.Point(4, 390);
this.txbPlayerName.Name = "txbPlayerName";
this.txbPlayerName.ReadOnly = true;
this.txbPlayerName.Size = new System.Drawing.Size(211, 22);
this.txbPlayerName.TabIndex = 2;
//
// panel4
//
this.panel4.Controls.Add(this.pctbMark);
this.panel4.Location = new System.Drawing.Point(221, 389);
this.panel4.Name = "panel4";
this.panel4.Size = new System.Drawing.Size(159, 142);
this.panel4.TabIndex = 1;
//
// panel3
//
this.panel3.Controls.Add(this.pictureBox1);
this.panel3.Location = new System.Drawing.Point(4, 4);
this.panel3.Name = "panel3";
this.panel3.Size = new System.Drawing.Size(379, 379);
this.panel3.TabIndex = 0;
//
// pctbMark
//
this.pctbMark.Location = new System.Drawing.Point(4, 4);
this.pctbMark.Name = "pctbMark";
this.pctbMark.Size = new System.Drawing.Size(152, 135);
this.pctbMark.SizeMode = System.Windows.Forms.PictureBoxSizeMode.StretchImage;
this.pctbMark.TabIndex = 0;
this.pctbMark.TabStop = false;
//
// pictureBox1
//
this.pictureBox1.Image = global::GameCaroLANServer.Properties.Resources.logocaro_png;
this.pictureBox1.Location = new System.Drawing.Point(4, 4);
this.pictureBox1.Name = "pictureBox1";
this.pictureBox1.Size = new System.Drawing.Size(372, 372);
this.pictureBox1.SizeMode = System.Windows.Forms.PictureBoxSizeMode.StretchImage;
this.pictureBox1.TabIndex = 0;
this.pictureBox1.TabStop = false;
//
// tmCoolDown
//
this.tmCoolDown.Tick += new System.EventHandler(this.TmCoolDown_Tick);
//
// menuStrip1
//
this.menuStrip1.ImageScalingSize = new System.Drawing.Size(20, 20);
this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.menuToolStripMenuItem});
this.menuStrip1.Location = new System.Drawing.Point(0, 0);
this.menuStrip1.Name = "menuStrip1";
this.menuStrip1.Size = new System.Drawing.Size(1231, 28);
this.menuStrip1.TabIndex = 2;
this.menuStrip1.Text = "menuStrip1";
//
// menuToolStripMenuItem
```

```
            //
            this.menuToolStripMenuItem.DropDownItems.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.newGameToolStripMenuItem,
            this.undoToolStripMenuItem,
            this.quitToolStripMenuItem});
            this.menuToolStripMenuItem.Name = "menuToolStripMenuItem";
            this.menuToolStripMenuItem.Size = new System.Drawing.Size(60, 24);
            this.menuToolStripMenuItem.Text = "Menu";
            //
            // newGameToolStripMenuItem
            //
            this.newGameToolStripMenuItem.Name = "newGameToolStripMenuItem";
            this.newGameToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control |
System.Windows.Forms.Keys.N)));
            this.newGameToolStripMenuItem.Size = new System.Drawing.Size(224, 26);
            this.newGameToolStripMenuItem.Text = "New Game";
            this.newGameToolStripMenuItem.Click += new System.EventHandler(this.NewGameToolStripMenuItem_Click);
            //
            // undoToolStripMenuItem
            //
            this.undoToolStripMenuItem.Name = "undoToolStripMenuItem";
            this.undoToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control |
System.Windows.Forms.Keys.Z)));
            this.undoToolStripMenuItem.Size = new System.Drawing.Size(224, 26);
            this.undoToolStripMenuItem.Text = "Undo";
            this.undoToolStripMenuItem.Click += new System.EventHandler(this.UndoToolStripMenuItem_Click);
            //
            // quitToolStripMenuItem
            //
            this.quitToolStripMenuItem.Name = "quitToolStripMenuItem";
            this.quitToolStripMenuItem.ShortcutKeys = ((System.Windows.Forms.Keys)((System.Windows.Forms.Keys.Control |
System.Windows.Forms.Keys.Q)));
```

```
            this.quitToolStripMenuItem.Size = new System.Drawing.Size(224, 26);
            this.quitToolStripMenuItem.Text = "Quit";
            this.quitToolStripMenuItem.Click += new System.EventHandler(this.QuitToolStripMenuItem_Click);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(1231, 724);
            this.Controls.Add(this.panel2);
            this.Controls.Add(this.pnlChessBoard);
            this.Controls.Add(this.menuStrip1);
            this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.MainMenuStrip = this.menuStrip1;
            this.Name = "Form1";
            this.Text = "Game Caro (LAN Server)";
            this.FormClosing += new System.Windows.Forms.FormClosingEventHandler(this.Form1_FormClosing);
            this.Shown += new System.EventHandler(this.Form1_Shown);
            this.panel2.ResumeLayout(false);
            this.panel2.PerformLayout();
            this.panel4.ResumeLayout(false);
            this.panel3.ResumeLayout(false);
            ((System.ComponentModel.ISupportInitialize)(this.pctbMark)).EndInit();
            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
            this.menuStrip1.ResumeLayout(false);
            this.menuStrip1.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }
        #endregion

        private System.Windows.Forms.Panel pnlChessBoard;
        private System.Windows.Forms.Panel panel2;
        private System.Windows.Forms.Label label2;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button btnLAN;
        private System.Windows.Forms.TextBox txbIP;
        private System.Windows.Forms.ProgressBar prcbCoolDown;
        private System.Windows.Forms.TextBox txbPlayerName;
        private System.Windows.Forms.Panel panel4;
        private System.Windows.Forms.PictureBox pctbMark;
        private System.Windows.Forms.Panel panel3;
        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Timer tmCoolDown;
        private System.Windows.Forms.MenuStrip menuStrip1;
        private System.Windows.Forms.ToolStripMenuItem menuToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem newGameToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem undoToolStripMenuItem;
        private System.Windows.Forms.ToolStripMenuItem quitToolStripMenuItem;
    }
}
```

d. ChessBoardManager.



```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GameCaroLANServer
{
    public class ChessBoardManager
    {
        Properties
        Initialize
        #region Methods
        public void DrawChessBroard()...
        private void btn_Click(object sender, EventArgs e)...
        public void EndGame()...
        public bool Undo()...

        private bool isEndGame(Button btn)...
        private Point GetChessPoint(Button btn)...
        private bool isEndHorizontal(Button btn)...

        private bool isEndVertical(Button btn)...

        private bool isEndPrimaryDiagonal(Button btn)...

        private bool isEndSub(Button btn)...

        private void Mark(Button btn)...

        private void ChangePlayer()...
        #endregion

    }
}
```

ChessBoardManager.cs

- Code detail:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GameCaroLANServer
{
    public class ChessBoardManager
    {
        #region Properties
        private Panel ChessBoard;

        public Panel ChessBoard1 { get => ChessBoard; set => ChessBoard = value; }
        public List<Player> Player { get => player; set => player = value; }
        public int CurrentPlayer { get => currentPlayer; set => currentPlayer = value; }
        public TextBox PlayerName1 { get => PlayerName; set => PlayerName = value; }
        public PictureBox PlayerMark { get => playerMark; set => playerMark = value; }
        public List<List<Button>> Matrix { get => matrix; set => matrix = value; }
        public Stack<PlayInfo> PlayTimeLine { get => playTimeLine; set => playTimeLine = value; }

        private List<Player> player;

        private int currentPlayer;
private TextBox PlayerName;

private PictureBox playerMark;

private List<List<Button>>matrix;

private event EventHandler endedGame;
public event EventHandler EndedGame
{
    add
    {
        endedGame += value;
    }
    remove
    {
        endedGame -= value;
    }
}

private event EventHandler playerMarked;
public event EventHandler PlayerMarked
{
    add
    {
        playerMarked += value;
    }
```

```
        remove
        {
            playerMarked -= value;
        }
    }
    private Stack<PlayInfo> playTimeLine;
    #endregion
    #region Initialize
    public ChessBoardManager(Panel ChessBoard, TextBox playerName, PictureBox mark)
    {
        this.ChessBoard = ChessBoard;
        this.PlayerName = playerName;
        this.PlayerMark = mark;
        this.Player = new List<Player>()
        {
            new Player("Mino", Image.FromFile(Application.StartupPath + "\\Resources\\teacher1.png")),
            new Player("Yoon", Image.FromFile(Application.StartupPath + "\\Resources\\teacher2.png"))
        };
    }
    #endregion
    #region Methods
    public void DrawChessBroard()
    {
        ChessBoard.Enabled = true;
        ChessBoard.Controls.Clear();
        PlayTimeLine = new Stack<PlayInfo>();
        currentPlayer = 0;
        ChangePlayer();
        Matrix = new List<List<Button>>();
        Button oldButton = 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(oldButton.Location.X + oldButton.Width, oldButton.Location.Y),
                    BackgroundImageLayout = ImageLayout.Stretch,
                    Tag = i.ToString()
                };
                btn.Click += btn_Click;
                ChessBoard.Controls.Add(btn);
                Matrix[i].Add(btn);
                oldButton = btn;
            }
            oldButton.Location = new Point(0, oldButton.Location.Y + Cons.CHESS_HEIGHT);
            oldButton.Width = 0;
            oldButton.Height = 0;
        }
    }
```

```
private 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();
    if (playerMarked != null)
        playerMarked(this, new EventArgs());
    if (isEndGame(btn))
    {
        EndGame();
    }
}
public void EndGame()
{
    if (endedGame != null)
        endedGame(this, new EventArgs());
}

public bool Undo()
{
    if (PlayTimeLine.Count <= 0)
        return false;
    PlayInfo oldPoint = PlayTimeLine.Pop();
```

```
        Button btn = matrix[oldPoint.Point.Y][oldPoint.Point.X];
        btn.BackgroundImage = null;

        if (PlayTimeLine.Count <= 0 )
        {
            CurrentPlayer = 0;
        }
        else
        {
            oldPoint = PlayTimeLine.Peek();
            CurrentPlayer = oldPoint.CurrentPlayer == 1 ? 0 : 1;
        }
        ChangePlayer();
        return true;
    }

    private bool isEndGame(Button btn)
    {
        return isEndHorizontal(btn) || isEndVertical(btn) || isEndPrimaryDiagonal(btn) || isEndSub(btn);
    }
    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;
    }
    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;
    }
```

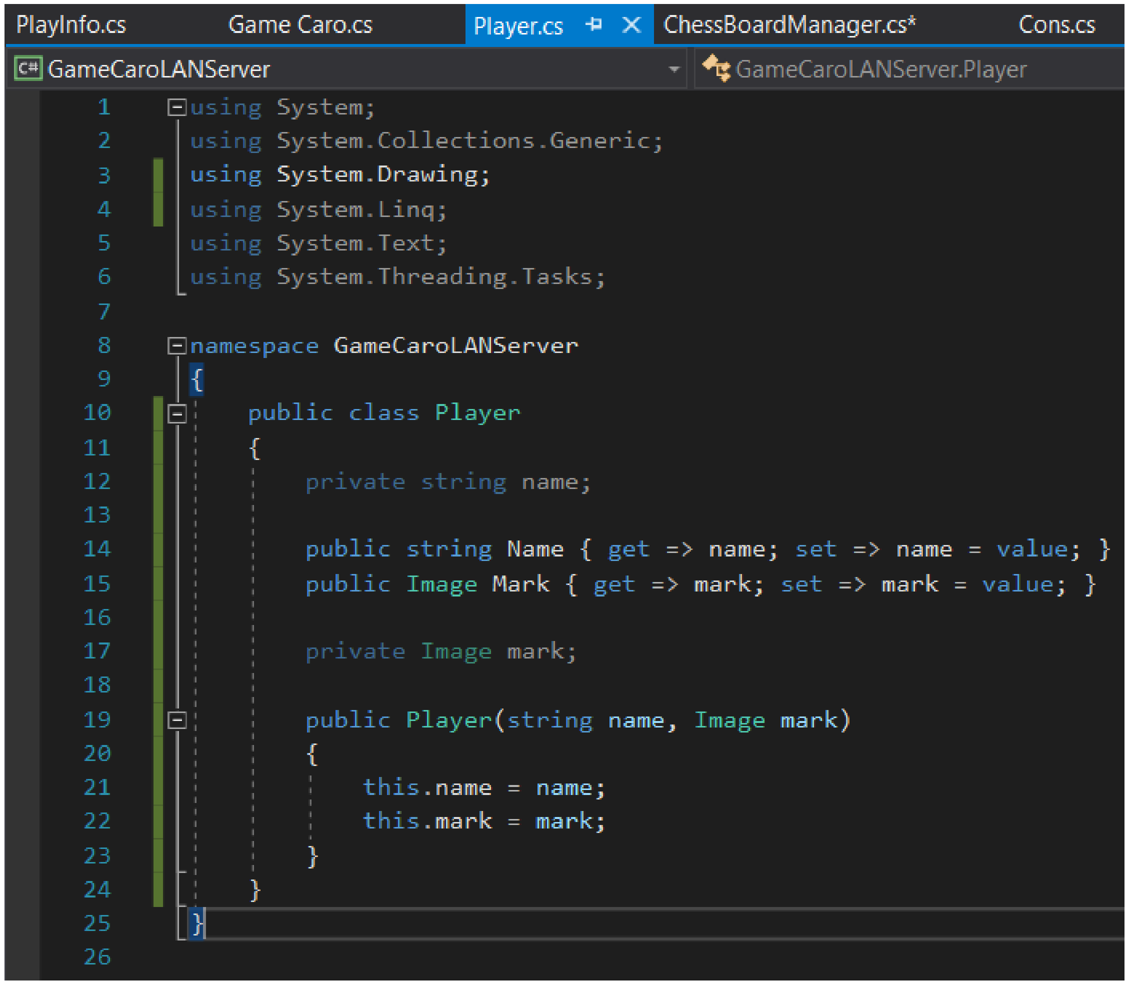
```
private bool isEndVertical(Button btn)
{
    Point point = GetChessPoint(btn);
    int countTop = 0;
    for (int i = point.Y; i >= 0; i--)
    {
        if (Matrix[i][point.X].BackgroundImage == btn.BackgroundImage)
        {
            countTop++;
        }
        else
            break;
    }
    int countBottom = 0;
    for (int i = point.Y + 1; i < Cons.CHESS_BOARD_HEIGHT; i++)
    {
        if (Matrix[i][point.X].BackgroundImage == btn.BackgroundImage)
        {
            countBottom++;
        }
        else
            break;
    }
    return countTop + countBottom == 5;
}

private bool isEndPrimaryDiagonal(Button btn)
```

```
{
    Point point = GetChessPoint(btn);
    int countTop = 0;
    for (int i = 0; i <= point.Y - i; i++)
    {
        if (point.X - i < 0 || point.Y - i < 0)
            break;
        if (Matrix[point.Y - i][point.X - i].BackgroundImage == btn.BackgroundImage)
        {
            countTop++;
        }
        else
            break;
    }
    int countBottom = 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)
        {
            countBottom++;
        }
        else
            break;
    }
    return countTop + countBottom == 5;
```

```
}

private bool isEndSub(Button btn)
{
    Point point = GetChessPoint(btn);
    int countTop = 0;
    for (int i = 0; i <= point.Y - i; 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)
        {
            countTop++;
        }
        else
            break;
    }
    int countBottom = 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)
        {
            countBottom++;
        }
        else
```

```
                break;
            }
            return countTop + countBottom == 5;
        }



        private void Mark(Button btn)
        {
            btn.BackgroundImage = Player[currentPlayer].Mark;
        }

        private void ChangePlayer()
        {
            PlayerName.Text = Player[currentPlayer].Name;

            PlayerMark.Image = Player[currentPlayer].Mark;
        }
        #endregion

    }
}
```

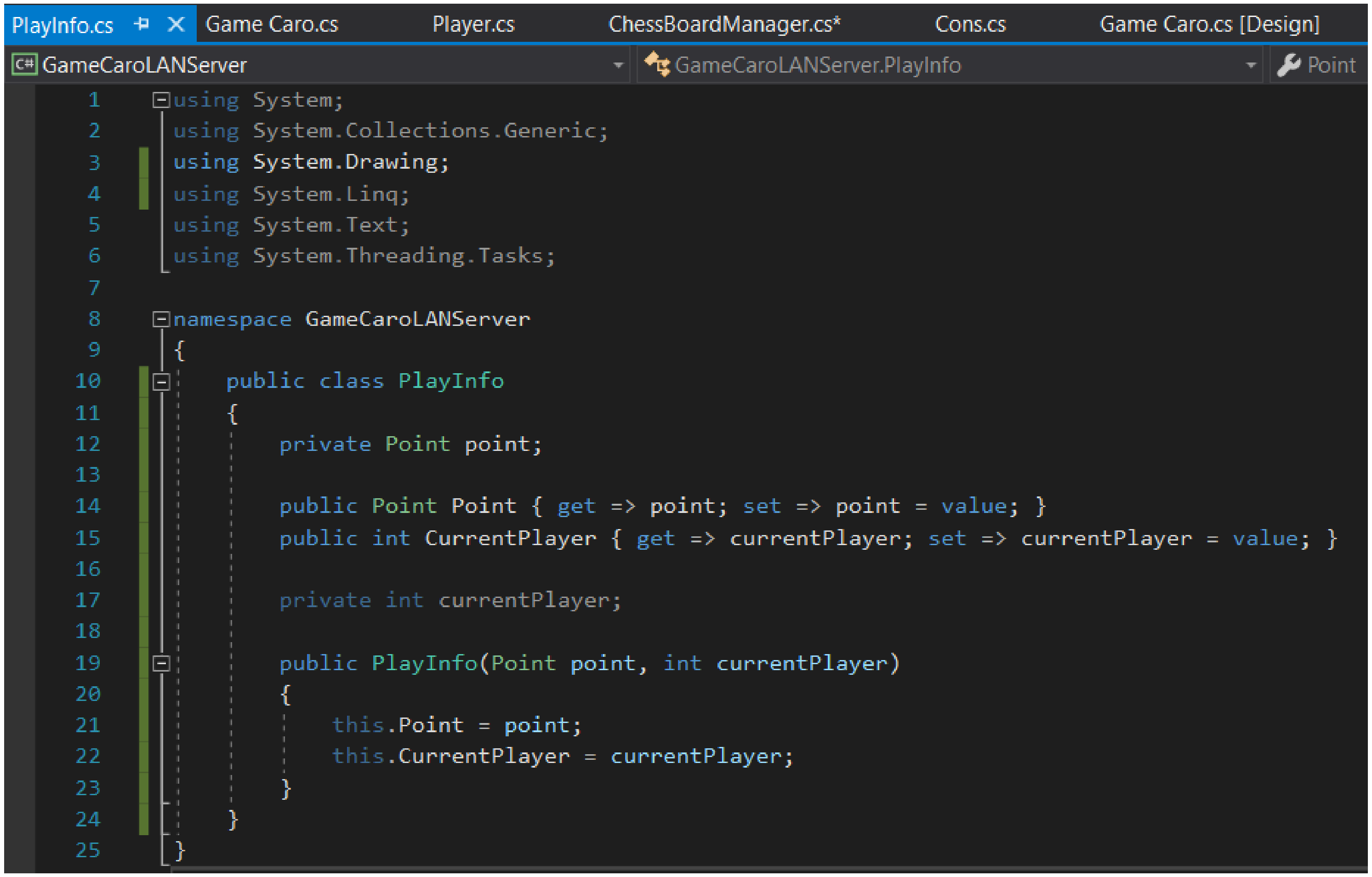
e. Player.



```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GameCaroLANServer
{
    public class Player
    {
        private string name;

        public string Name { get => name; set => name = value; }
        public Image Mark { get => mark; set => mark = value; }

        private Image mark;

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

Player.cs

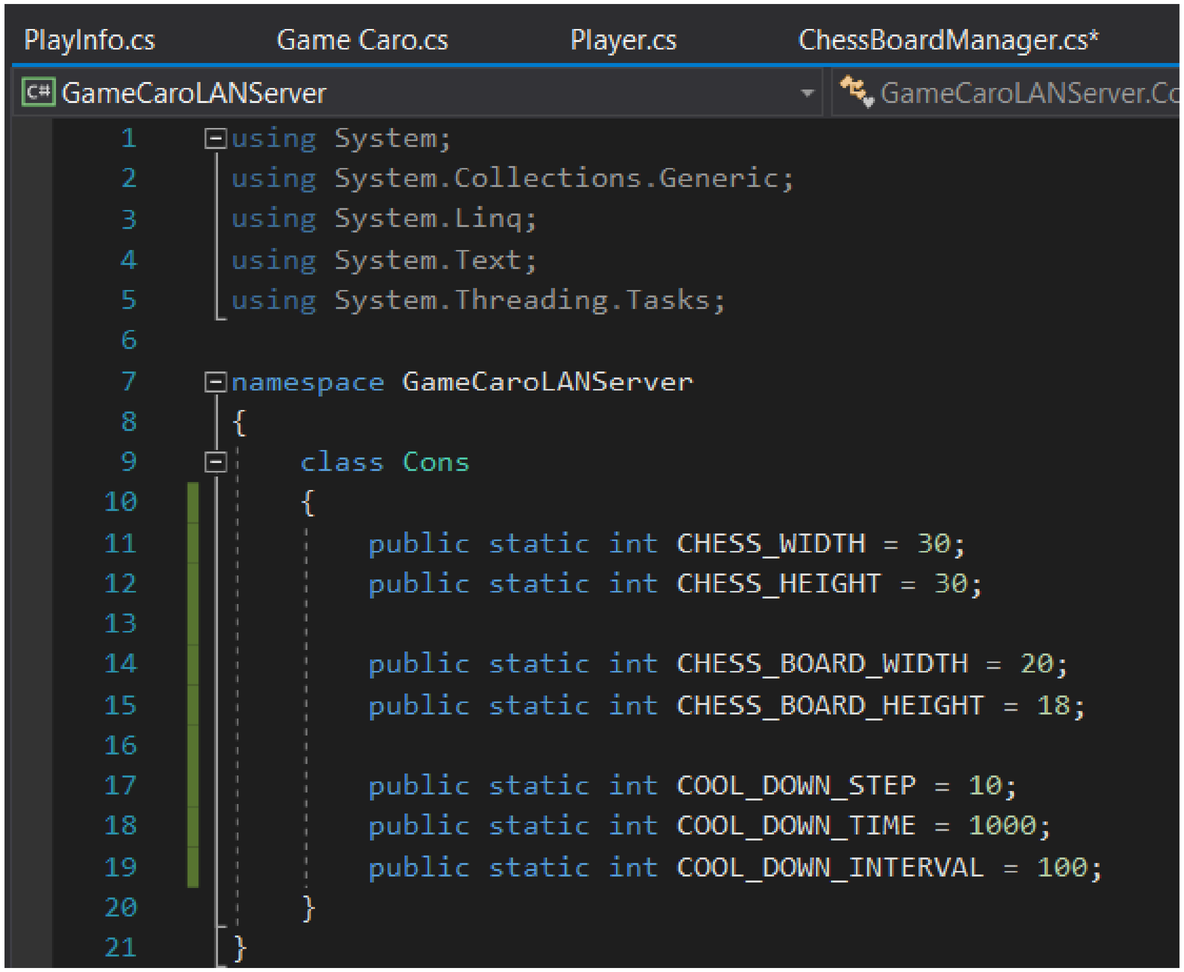
f. PlayInfo.



```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GameCaroLANServer
{
    public class PlayInfo
    {
        private Point point;

        public Point Point { get => point; set => point = value; }
        public int CurrentPlayer { get => currentPlayer; set => currentPlayer = value; }

        private int currentPlayer;

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

PlayInfo.cs

g. Cons.

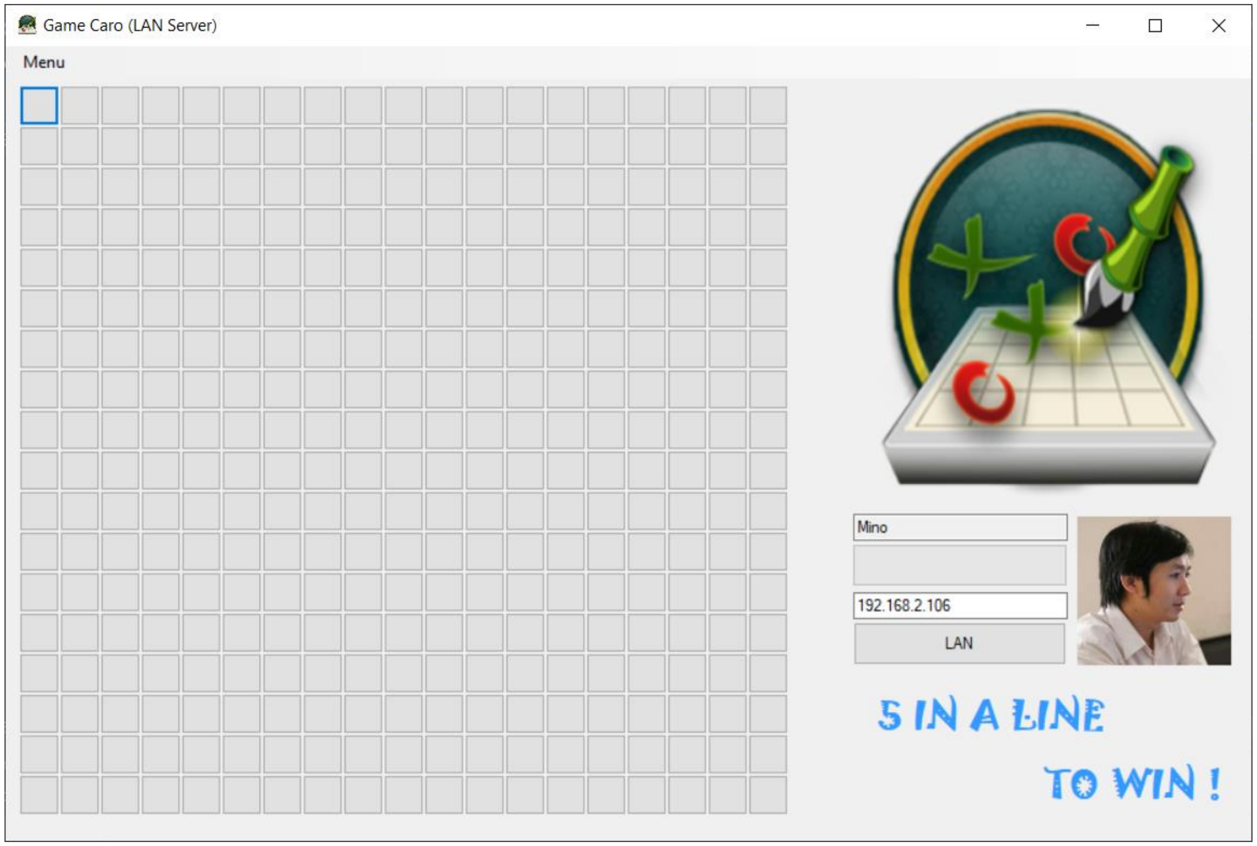

PlayInfo.cs Game Caro.cs Player.cs ChessBoardManager.cs*

GameCaroLANServer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

        public static int CHESS_BOARD_WIDTH = 20;
        public static int CHESS_BOARD_HEIGHT = 18;

        public static int COOL_DOWN_STEP = 10;
        public static int COOL_DOWN_TIME = 1000;
        public static int COOL_DOWN_INTERVAL = 100;
    }
}
```

Cons.cs

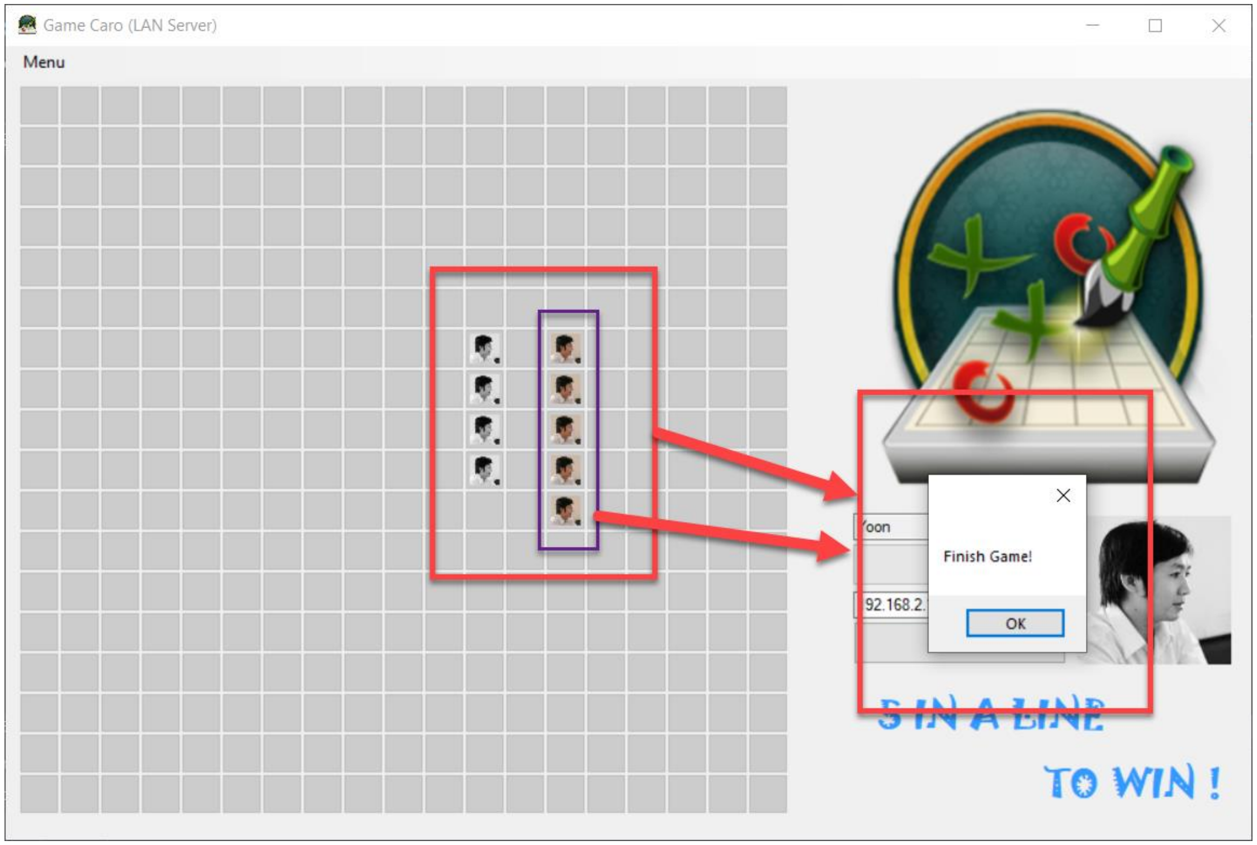
❖ **Debug**



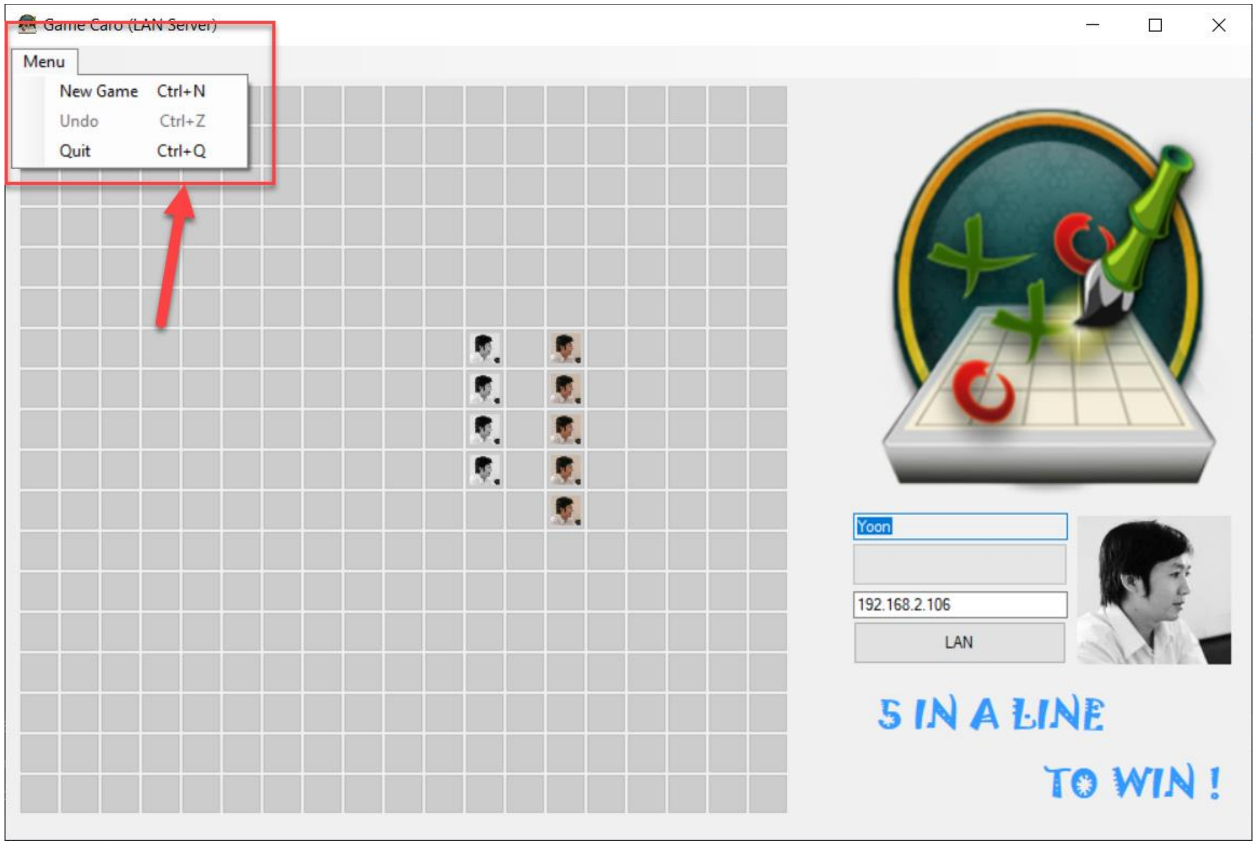
Mino

192.168.2.106

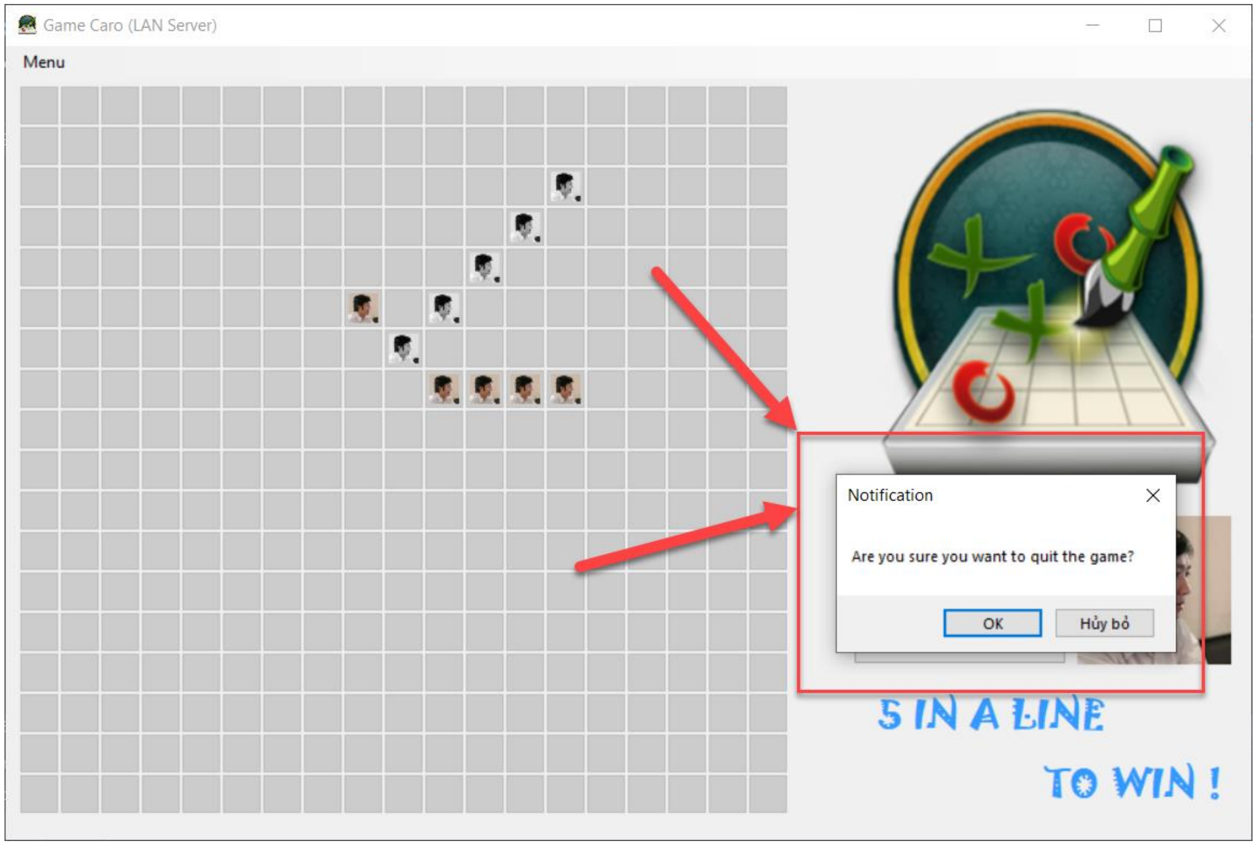
Images 1.


Game Caro (LAN Server)
Menu
Yoon
92.168.2.
Finish Game!
OK
5 IN A LINE
TO WIN !

Images 2.

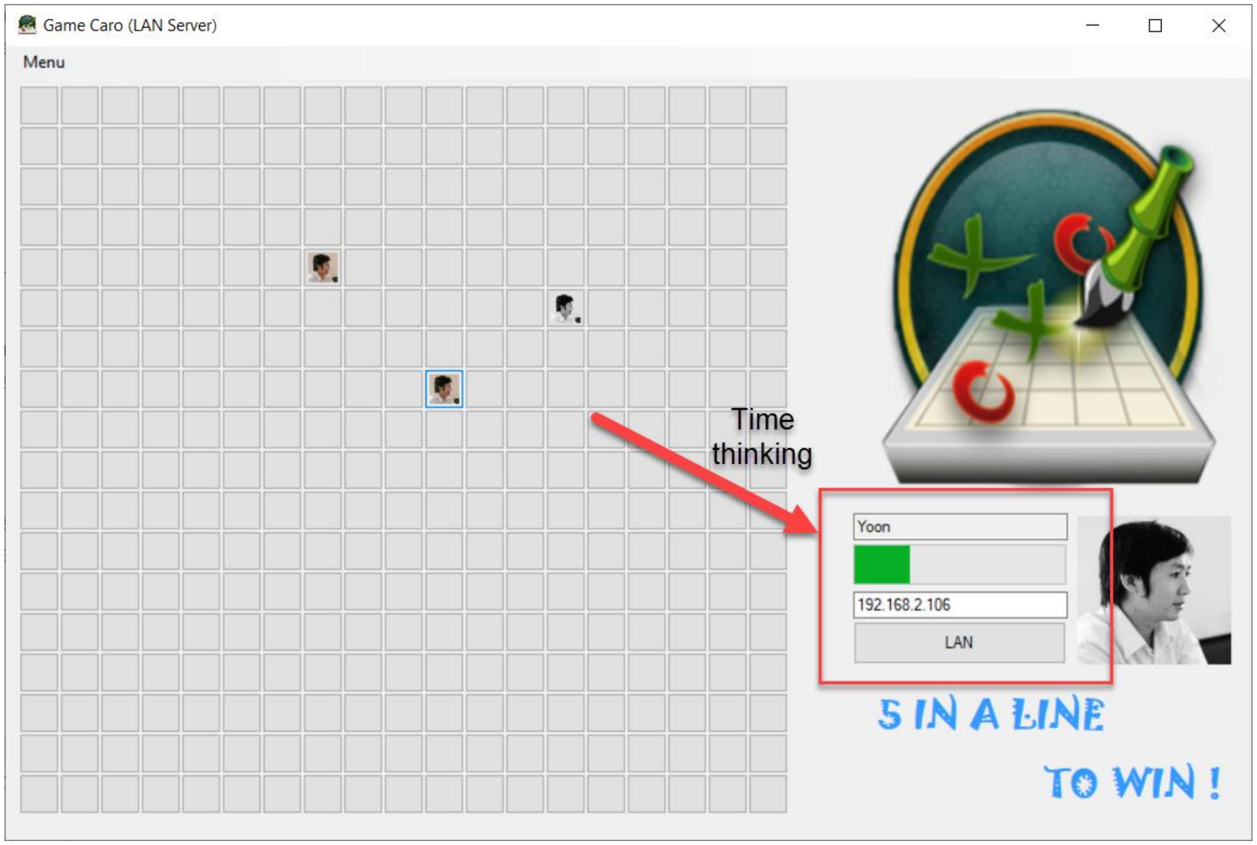

Game Caro (LAN Server)
Menu
New Game Ctrl+N
Undo Ctrl+Z
Quit Ctrl+Q
Yoon
192.168.2.106
LAN
5 IN A LINE
TO WIN !

Images 3.



Are you sure you want to quit the game?

Images 4.



Images 5.

II. Evaluate the use of an IDE for development of applications contrasted with not using an IDE.

- In most cases, an IDE is a text editor with some extra bits added on to help you write code more easily. The text editor is at the core of the IDE, and works more or less like you'd expect. Many will highlight simple syntax errors on the fly, so you don't have the experience of having a simple typo ruin hours of coding. They also often have an "autocomplete" feature, so that if you type system.math. and pause, the IDE will present you with a list of the various math functions available, so you don't need to remember whether the square root function is sqrt() or squareroot().

- Just about every IDE has a build automator of some kind that compiles the code, and builds an executable program by adding in any necessary libraries, some of which you may not even be aware of. Those same tools make it easier to organize and include any optional libraries, or ones you've created yourself.

- The feature of most IDEs that you'd be spending the most time with, though, is the debugger. Depending on the IDE, it's a suite of tools that will find errors in your code before it compiles, and then (hopefully) point them out to you so that you can fix them easily. That's an oversimplification, and it's far from foolproof.

- You may also hear stories about "real programmers" who don't use IDEs, but type all of their code perfectly the first time, from beginning to end, perhaps even in hexadecimal. Even if you don't, you'll probably hear from somebody who thinks that IDEs are overkill, and you can do everything you need with vi or emacs (which will then start an argument about whether vim or emacs add-ons constitute an IDE, and you don't want any part of that; just back away slowly). Speaking very broadly, programmers who work in scripting languages, or languages that evolved from scripting languages, are more likely to regard an IDE as optional.

- The programs you'll be writing at this stage will generally be pretty short, and you won't miss most of the features of a full IDE. You'll be able to shorten the feedback loop between code and results if you don't have to worry about a tool you're not conversant with yet. The period before you're ready for an IDE may be very short, a matter of days or weeks, depending on how quickly you're learning. As a rule of thumb, once your code goes beyond one file, you should invest some time in learning an IDE.

- Depending on the language, you may find it difficult to get any instructions on how to code without an IDE, or you may find the command-line interface for the compiler to be more obscure than a visual IDE. The bottom line is that when you're learning to program, the tool should be a help, not a roadblock.

C. Determine the debugging process and explain the importance of a coding standard.

I. Explain the debugging process and explain the debugging facilities available in the IDE.

- Definition: Debugging is the process of detecting and removing of existing and potential errors (also called as ‘bugs’) in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another. Sometimes it takes more time to debug a program than to code it.

- Description: To debug a program, user has to start with a problem, isolate the source code of the problem, and then fix it. A user of a program must know how to fix the problem as knowledge about problem analysis is expected. When the bug is fixed, then the software is ready to use. Debugging tools (called debuggers) are used to identify coding errors at various development stages. They are used to reproduce the conditions in which error has occurred, then examine the program state at that time and locate the cause. Programmers can trace the program execution step-by-step by evaluating the value of variables and stop the execution wherever required to get the value of variables or reset the program variables. Some programming language packages provide a debugger for checking the code for errors while it is being written at run time.

<u>The debugging process:</u>

- Reproduce the problem.
- Describe the bug. Try to get as much input from the user to get the exact reason.
- Capture the program snapshot when the bug appears. Try to get all the variable values and states of the program at that time.
- Analyse the snapshot based on the state and action. Based on that try to find the cause of the bug.
- Fix the existing bug, but also check that any new bug does not occur.

- A good IDE will have a graphical debugger, so instead of having to remember line numbers, put breaks on, or the names of variables to watch. You can click on the lines you want to on, and click on the variables to watch. Essentially an easy to use point and click interface, is far easier to use than a command line interface. A good IDE will enable you to also see the context for some code - you can scroll up and down and even look at other source code segments in other files, in order to understand how the code being debugged is called and what it will call next. The more information you have, and the easier it is to get that information the easier the process will be.

- A great IDE helps debugging mostly via bookmarking the code. For example, you bookmark a line and the code is stopped at this line when in debug mode. At that time, you can see which are the values for all your local variables, which is the local calls stack. You can also add some Watches and watch some evaluated expressions that are changed dynamically. You can then proceed line by line and see how all these are affected, without waiting for the whole process to end. In advanced IDEs such as Visual Studio, you can even watch parallel stacks (used in parallel programming).

- A good ide will give you conditional breakpoints, variable inspection at various levels of the call stack. Also allows you to refactor the code to make debugging easier. Truly awesome ides allow you to step back one or two steps from a crash to inspect the state before the problem.

- Besides, IDE (as in Integrated Development Environment) combines all the steps necessary in developing software in one place,including debugging. And as part of that, it gives you ready access to both the source code in the editor part as well as the machine code that is produced by the compiler.

II. Outline the coding standard you have used in your code.

<u>The coding standards that I use in my code are:</u>

- Consistent naming diagram.
- Code group: I group tasks in blocks.
- No write comment code when unnecessarily.
- Indentation consistent.
- Write code fully and clearly .

III. Evaluate how the debugging process can be used to help develop more secure, robust applications.

- Test-driven Development means building automated unit and regression tests early in the development cycle, and the maintaining and running them throughout. This is effectively investing in proactive debugging, and it produces more robust software applications. From a security perspective, automated tests can be based on ‘misuse’ cases, rather than use cases, but the principle is the same. Extreme, or Pair Programming is also a valuable way of debugging code proactively, especially with one of the paired programmers has a security background.

- Debugging is the process of finding the cause of and fixing identified mistakes in software. These bugs are found either by testing or by users in the field. It is what you do when there is a known problem. As such, debugging cannot help develop more secure applications but is essential to improving the robustness of applications. Because debugging is a reactive process, its not a good way to improve security or robustness. Testing is better, but you can’t test an application into perfection. The best way is to have good design and development processes and to hire software developers and architects who really know what they are doing.

- It is better to set up your project so that it uses a test suite that is compatible with the programming tools you are using. You can write unit tests, which are short programs that define what each part of your application are supposed to do, and then you run the test suite against your existing code and see if all the tests pass. If not, you either have exposed a bug that you need to fix, or your test suite is not quite right.

IV. Critically evaluate why a coding standard is necessary in a team as well as for the individual.

- A coding standard ensures that all developers writing the code in a particular language write according to the guidelines specified. This makes the code easy to understand and provides consistency in the code.

- One of the most essential factors in a software system is the consistency of the coding standard. This is because it positively impacts the quality of the system. While using a software system, you need to ensure that the guidelines do not contradict each other. The source code that uses the standard should also be in harmony with the standard. The completed source code should indicate as if the code has been written by a single developer in a single session.

- Benefits of coding standards: Increases Efficiency, Minimize the Risk of Project Failure, Reduces Complexity, Maintenance becomes easy, Correction of bugs, A comprehensive view, Cost saving.

→ ***Coding standards provide clarity to the purpose of the code. A neat well laid out code with optimum commenting is easy to comprehend and enhance. They make it easier to be used for teamwork as well.***

D. Reference.

https://study.com/academy/answer/provide-explanations-of-what-procedural-object-orientated-and-event-driven-paradigms-are-include-characteristics-and-the-relationship-between-them.html

https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment

https://study.com/academy/answer/critically-evaluate-the-source-code-of-an-application-which-implements-the-programming-paradigms-in-terms-of-the-code-structure-and-characteristics.html

http://radar.oreilly.com/2014/01/to-ide-or-not-to-ide.html

https://www.quora.com/How-does-IDE-help-in-the-debugging-procedure

https://economictimes.indiatimes.com/definition/debugging

https://www.quora.com/How-can-the-debugging-process-be-used-to-help-develop-more-secure-robust-applications

https://webguruz.in/the-significance-of-coding-standards/