TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

BÁO CÁO MÔN HỌC PROJECT I

Giảng viên hướng dẫn: Thân Quang Khoát

Đề tài*: Tìm hiểu ngôn ngữ C#, Winform và lập trình game cờ Caro kết nối mạng LAN*

Sinh viên: Nguyễn Kiên Trung

Mã số sinh viên: 20173421

Lớp: Khoa học máy tính 03

Mục lục

Lời mở đầu

Hiện nay, với sự ra đời và phát triển của nhiều ngôn ngữ lập trình, C# (C Sharp) dù đã ra đời từ rất lâu nhưng vẫn giữ được sự phổ biến và hữu dụng. C# với sự hỗ trợ mạnh mẽ của .NET Framework giúp cho việc tạo một ứng dụng Windows Forms trở nên rất dễ dàng.

Trong thời đại này, Internet là một thứ tất yếu trong cuộc sống của mỗi con người. Con người sử dụng Internet để kết nối với nhau, học tập trao đổi tri thức hoặc để giải trí,... Từ những lí do trên, em có ý tưởng lập trình một trò chơi rất quen thuộc với mỗi chúng ta, đó là trò chơi cờ caro, sử dụng ngôn ngữ lập trình C# kết nối trong mạng LAN để hai người có thể đấu với nhau.

Em xin cảm ơn các thầy cô trường Đại học Bách Khoa Hà Nội, các thầy cô viện Công nghệ thông tin và Truyền thông đã truyền đạt kiến thức, giúp đỡ em trong quá trình học tập. Em xin đặc biệt cảm ơn thầy giáo Thân Quang Khoát đã hướng dẫn, giúp đỡ em hoàn thành môn học này.

1 Tìm hiểu ngôn ngữ lập trình C#

1.1 Tổng quan

C# (hay C sharp) là một ngôn ngữ lập trình đơn giản, được phát triển bởi đội ngũ kỹ sư của Microsoft vào năm 2000, trong đó người dẫn đầu là Anders Hejlsberg và Scott Wiltamuth.

C# là ngôn ngữ lập trình hiện đại, hướng đối tượng và nó được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.

C# được thiết kế cho Common Language Infrastructure (CLI), mà gồm Executable Code và Runtime Environment, cho phép chúng ta sử dụng các ngôn ngữ high-level đa dạng trên các nền tảng và cấu trúc máy tính khác nhau.

C# với sự hỗ trợ mạnh mẽ của .NET Framework giúp cho việc tạo một ứng dụng Windows Forms hay WPF (Windows Presentation Foundation), . . . trở nên rất dễ dàng.

1.2 Đặc trưng của C#

Các đặc điểm để làm cho C# là ngôn ngữ lập trình chuyên nghiệp được sử dụng rộng rãi:

C# là ngôn ngữ đơn giản

Như ta đã biết thì ngôn ngữ C# dựng trên nền tảng C++ và Java nên ngôn ngữ C# khá đơn giản. Nếu chúng ta thân thiện với C và C++ hoặc thậm chí là Java, chúng ta sẽ thấy C# khá giống về diện mạo, cú pháp, biểu thức, toán tử và những chức năng khác được lấy trực tiếp từ ngôn ngữ C và C++, nhưng nó đã được cải tiến để làm cho ngôn ngữ đơn giản hơn. Một vài trong các sự cải tiến là loại bỏ các dư thừa, hay là thêm vào những cú pháp thay đổi.

C# là ngôn ngữ hiện đại

Một vài khái niệm khá mới mẻ khá mơ hồ với các bạn vừa mới học lập trình, như xử lý ngoại lệ, những kiểu dữ liệu mở rộng, bảo mật mã nguồn..v..v... Đây là những đặc tính được cho là của một ngôn ngữ hiện đại cần có. Và C# chứa tất cả các đặt tính ta vừa nêu trên.

C# là một ngôn ngữ lập trình thuần hướng đối tượng

Lập trình hướng đối tượng(tiếng Anh: Object-oriented programming, viết tắt: OOP) là một phương pháp lập trình có 4 tính chất. Đó là tính trừu tượng (abstraction), tính đóng gói (encapsulation), tính đa hình (polymorphism) và tính kế thừa (inheritance). C# hỗ trợ cho chúng ta tất cả những đặc tính trên.

C# là một ngôn ngữ ít từ khóa

C# là ngôn ngữ sử dụng giới hạn những từ khóa (gồm khoảng 80 từ khóa và hơn mười kiểu dữ liệu xây dựng sẵn). Nếu bạn nghĩ rằng ngôn ngữ có càng nhiều từ khóa thì sẽ càng mạnh mẽ hơn. Điều này không phải sự thật, lấy ví dụ ngôn ngữ C# làm điển hình nhé. Nếu bạn học sâu về C# bạn sẽ thấy rằng ngôn ngữ này có thể được sử dụng để làm bất cứ nhiệm vụ nào.

Ngoài những đặc điểm trên thì còn một số ưu điểm nổi bật của C#:

- C# có cấu trúc khá gần gũi với các ngôn ngữ lập trình truyền thống, nên cũng khá dễ dàng tiếp cận và học nhanh với C#.
- C# có thể biên dịch trên nhiều nền tảng máy tính khác nhau.
- C# được xây dựng trên nền tảng của C++ và Java nên nó được thừa hưởng những ưu điểm của ngôn ngữ đó.
- C# là một phần của .NET Framework nên được sự chống lưng khá lớn đến từ bộ phận này.
- C# có IDE Visual Studio cùng nhiều plug-in vô cùng mạnh mẽ.

1.3 Cơ bản về ngôn ngữ lập trình C#

1.3.1 Kiểu dữ liệu

Các biến kiểu giá trị có thể được gán một giá trị một cách trực tiếp. Chúng được kế thừa từ lớp **System.ValueType**.

Những kiểu dữ liệu cơ bản:

Kiểu	Biểu diễn	Dãy giá trị	Giá trị mặc định
bool	Giá trị Boolean	True hoặc False	False
byte	Kiểu unsigned integer (8 bit)	0 tới 255	0
char	Kiểu Unicode character (16 bit)	U +0000 tới U +ffff	'\0'
decimal	Kiểu thập phân (128 bit)	$(-7.9 \times 10^{28}$ tới $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$	0.0M
double	Kiểu double (64 bit)	$(+/-)5.0 \times 10^{-324}$ tới $(+/-)1.7 \times 10^{308}$	0.0D
float	Kiểu float (32 bit)	-3.4×10^{38} tới $+ 3.4 \times 10^{38}$	0.0F
int	Kiểu integer (32 bit)	-2,147,483,648 tới 2,147,483,647	0
long	Kiểu signed integer (64 bit)	-9,223,372,036,854,775,808 tới 9,223,372,036,854,775,807	0L
sbyte	Kiểu signed integer (8 bit)	-128 tới 127	0
short	Kiểu signed integer (16 bit)	-32,768 tới 32,767	0
uint	Kiểu unsigned integer (32 bit)	0 tới 4,294,967,295	0
ulong	Kiểu unsigned integer (64 bit)	0 tới 18,446,744,073,709,551,615	0
ushort	Kiểu unsigned integer (16 bit)	0 tới 65,535	0

Kiểu tham chiếu trong C#:

Kiểu tham chiếu không chứa dữ liệu thực sự được lưu giữ trong một biến, nhưng chúng chứa một tham chiếu tới các biến. Nói cách khác, chúng tham chiếu tới một vị trí bộ nhớ. Việc sử dụng nhiều biến, thì kiểu tham chiếu có thể tham chiếu tới tới một vị trí bộ nhớ. Nếu dữ liệu trong vị trí bộ nhớ bị thay đổi bởi một trong số các biến, thì biến khác tự động phản ánh sự thay đổi về giá trị này. Ví dụ các kiểu tham chiếu có sẵn trong C# là: **object**, **dynamic,** và **string**.

- Kiểu object: Kiểu object là lớp cơ sở cơ bản cho tất cả kiểu dữ liệu trong C# Common Type System (CTS). Object là một alias cho lớp System.Object. Các kiểu object có thể được gán giá trị của bất kỳ kiểu, kiểu giá trị, kiểu tham chiếu, kiểu tự định nghĩa (user-defined) khác. Tuy nhiên, trước khi gán các giá trị, nó cần một sự chuyển đổi kiểu. Khi một kiểu giá trị được chuyển đổi thành kiểu object, nó được gọi là boxing và ngược lại, khi một kiểu object được chuyển đổi thành kiểu giá trị, nó được gọi là unboxing.
- Kiểu dynamic :Bạn có thể lưu giữ bất kỳ kiểu giá trị nào trong biến kiểu dữ liệu dynamic. Việc kiểm tra các kiểu biến này diễn ra tại run time. Kiểu dynamic là tương tự với kiểu object, ngoại trừ việc kiểm tra cho các biến kiểu object diễn ra tại compile time, trong khi việc kiểm tra các biến kiểu dynamic diễn ra tại run time.
- Kiểu string : Kiểu string trong C# cho phép bạn gán bất kỳ giá trị chuỗi nào cho một biến. Kiểu string là một alias cho lớp System.String. Nó kế thừa từ kiểu object. Giá trị cho một kiểu string có thể được gán bởi sử dụng các hằng chuỗi.

Kiểu con trỏ trong C#:

Các biến kiểu con trỏ lưu giữ địa chỉ bộ nhớ của kiểu khác. Các con trỏ trong C# có khả năng như con trỏ trong C hoặc C++.

1.3.2 Câu lệnh if, else, switch

Tương tự như C, C++, những câu lệnh này kiểm tra 1 điều kiện nào đó trước khi thực hiện khối lệnh bên trong:

Lệnh	Miêu tả
Lệnh if trong C#	Một lệnh **if** bao gồm một biểu thức logic theo sau bởi một hoặc nhiều lệnh khác.
Lệnh if...else trong C#	Một lệnh **if** có thể theo sau bởi một lệnh **else** (tùy ý: có hoặc không), mà có thể được thực hiện khi biểu thức logic có giá trị false.
Lồng các lệnh if trong C#	Bạn có thể sử dụng lệnh **if** hoặc lệnh **else if** bên trong lệnh **if** hoặc **else if** khác
Lệnh switch trong C#	Lệnh **switch** cho phép kiểm tra điều kiện của một biến trước khi thực thi các lệnh
Lồng các lệnh switch trong C#	Bạn có thể sử dụng một lệnh **switch** bên trong một lệnh **switch** khác

Toán tử ? : trong C#

Toán tử điều kiện **? :** có thể được dùng để thay thế cho lệnh **if...else**.

Ví dụ: Exp1? Exp2 : Exp3

Trong đó Exp1, Exp2 và Exp3 là các biểu thức. Giá trị của biểu thức Exp1 trước dấu ? có giá trị **true**, Exp2 được thực hiện, và giá trị của nó là giá trị của biểu thức. Nếu Exp1 là **false** thì Exp3 được thực hiện và giá trị của nó là giá trị của biểu thức.

1.3.3 Vòng lặp

Có một tình huống mà bạn cần phải thực hiện một đoạn code một vài lần. Nhìn chung, các câu lệnh được thực hiện một cách tuần tự. Câu lệnh đầu tiên của hàm được thực hiện trước, sau đó đến câu thứ 2 và tiếp tục. Tương tự C, C++, ngôn ngữ lập trình C# hỗ trợ những câu lệnh điều khiển sau:

Kiểu vòng lặp	Miêu tả
Vòng lặp while trong C#	Lặp lại một hoặc một nhóm các lệnh trong khi điều kiện đã cho là đúng. Nó kiểm tra điều kiện trước khi thực hiện thân vòng lặp.
Vòng lặp for trong C#	Thực thi một dãy các lệnh nhiều lần và tóm tắt đoạn code mà quản lý biến vòng lặp.
Vòng lặp do...while trong C#	Giống lệnh while, ngoại trừ ở điểm là nó kiểm tra điều kiện ở cuối thân vòng lặp.
Lồng các vòng lặp trong C#	Bạn có thể sử dụng một hoặc nhiều vòng lặp trong các vòng lặp while, for hoặc do..while khác.

Cùng với các câu lệnh để thực hiện vòng lặp là các lệnh để điều khiển vòng lặp: **break, continue**

Lệnh điều khiển	Miêu tả
Lệnh break trong C#	Kết thúc **vòng lặp** hoặc lệnh **switch** và chuyển sang thực thi vòng lặp hoặc lệnh switch ngay sau nó.
Lệnh continue trong C#	Khi gặp lệnh này thì chương trình sẽ bỏ qua các câu lệnh ở dưới nó (trong cùng một câu lệnh lặp) để thực hiện vòng lặp mới.

1.3.4 Tính đóng gói trong C#

Tính đóng gói, trong phương pháp lập trình hướng đối tượng, ngăn cản việc truy cập tới chi tiết của trình triển khai (Implementation Detail).

Tính trừu tượng và tính đóng gói là hai đặc điểm có liên quan với nhau trong lập trình hướng đối tượng. Tính trừu tượng cho phép tạo các thông tin liên quan có thể nhìn thấy và tính đóng gói cho lập trình viên khả năng triển khai độ trừu tượng đã được kế thừa.

Tính đóng gói được triển khai bởi sử dụng **Access Specifier**. Một Access Specifier định nghĩa phạm vi và tính nhìn thấy của một thành viên lớp. C# hỗ trợ các Access Specifier sau:

- Public: cho phép một lớp trưng bày các biến thành viên và các hàm thành viên của nó tới các hàm và đối tượng khác. Bất kỳ thành viên public nào trong C# có thể được truy cập từ bên ngoài lớp đó.
- Private: cho phép một lớp ẩn các biến thành viên và các hàm thành viên của nó với các hàm và đối tượng khác. Chỉ có các hàm trong cùng lớp đó có thể truy cập tới các thành viên private. Ngay cả khi một Instance của một lớp cũng không thể truy cập các thành viên private của nó.
- Protected: cho phép một lớp con truy cập các biến thành viên và các hàm thành viên của lớp cơ sở của nó. Cách này giúp triển khai tính kế thừa.
- Internal: cho phép một lớp trưng bày các biến thành viên và các hàm thành viên của nó tới các hàm và đối tượng khác trong Assembly hiện tại. Nói cách khác, bất kỳ thành viên nào với Internal Access Specifier trong C# có thể được truy cập từ bất kỳ lớp hoặc phương thức được định nghĩa bên trong ứng dụng mà thành viên đó được định nghĩa.
- Protected internal: cho phép một lớp ẩn các biến thành viên và các hàm thành viên của nó với các hàm và đối tượng khác, ngoại trừ một lớp con bên trong cùng ứng dụng đó. Điều này cũng được sử dụng trong khi triển khai tính kế thừa trong C#.

1.3.5 Phương thức (Hàm)

Một phương thức là một nhóm lệnh cùng nhau thực hiện một tác vụ. Mỗi chương trình C# có ít nhất một lớp với một phương thức là Main.

Để sử dụng một phương thức trong C#, bạn cần định nghĩa phương thức và gọi phương thức

Định nghĩa phương thức trong C#:

<Access Specifier> <Kiểu_trả_về> <tên_phương_thức>(danh_sách_tham_số)
{
phần thân phương thức
}

- **Access Specifier**: Định nghĩa tính nhìn thấy của một biến hoặc một phương thức với lớp khác.
- **Kiểu_trả_về**: Một phương thức có thể trả về một giá trị. Kiểu trả về là kiểu dữ liệu của giá trị mà phương thức trả về. Nếu phương thức không trả về bất kỳ giá trị nào, thì kiểu trả về là **void**.
- **tên_phương_thức**: Tên phương thức là một định danh duy nhất và nó là phân biệt kiểu chữ. Nó không thể giống bất kỳ định danh nào khác đã được khai báo trong lớp đó.
- **danh_sách_tham_số**: Danh sách tham số được bao quanh trong dấu ngoặc đơn, các tham số này được sử dụng để truyền và nhận dữ liệu từ một phương thức. Danh sách tham số liên quan tới kiểu, thứ tự, và số tham số của một phương thức. Các tham số là tùy ý, tức là một phương thức có thể không chứa tham số nào.
- **phần thân phương thức**: Phần thân phương thức chứa tập hợp các chỉ thị cần thiết để hoàn thành hoạt động đã yêu cầu.

Gọi phương thức trong C#:

Bạn có thể gọi một phương thức bởi sử dụng tên của phương thức đó kèm theo danh sách tham số để phương thức đó được thực hiện.

Ví dụ:

Định nghĩa phương thức:

Public int Max(int a, int b){
If(a > b) return a;
Else retun b;

Gọi phương thức:

Max(a, b);

1.3.6 Enum

Một Enumeration (liệt kê) là một tập hợp các hằng số nguyên được đặt tên. Một kiểu enum được khai báo bởi sử dụng từ khóa **enum** trong C#.

Các kiểu liệt kê trong C# là kiểu dữ liệu giá trị. Nói cách khác, kiểu liệt kê chứa các giá trị của nó và không thể kế thừa hoặc không thể truyền tính kế thừa.

Khai báo biến enum trong C#:

Cú pháp chung để khai báo một Enumeration trong C# là:

```
enum <tên_enum>
{
  danh_sách_enum
};
```

- *tên_enum* xác định tên kiểu liệt kê.
- *danh_sách_enum* là danh sách các định danh được phân biệt nhau bởi dấu phẩy.

1.3.7 Lớp (Class)

Khi bạn định nghĩa một lớp (class) trong C#, như bạn định nghĩa cho một kiểu dữ liệu. Điều này không thực sự định nghĩa bất kỳ dữ liệu nào, nhưng nó định nghĩa ý nghĩa của tên lớp đó. Tức là, một đối tượng của lớp đó gồm những cái gì, các hoạt động nào có thể được thực hiện trên đối tượng đó. Các đối tượng là instance (sự thể hiện) của một lớp. Các phương thức và các biến mà cấu tạo nên một lớp được gọi là các thành viên của lớp đó.

Định nghĩa một Class trong C#:

Một định nghĩa lớp trong C# bắt đầu với từ khóa class được theo sau bởi tên lớp và phần thân lớp được bao quanh bởi các dấu ngoặc ôm. Dưới đây là form chung của một định nghĩa lớp trong C#:

```
<access specifier> class  tên_lớp
{
  // các biến thành viên
  <access specifier> <kiểu_dữ_liệu> biến1;
  <access specifier> <kiểu_dữ_liệu> biến2;
  ...
  <access specifier> <kiểu_dữ_liệu> biếnN;

  // các phương thức thành viên
  <access specifier> <kiểu_trả_về> tên_phương_thức1(danh_sách_tham_số)
  {
    // phần thân phương thức
  }
  <access specifier> <kiểu_trả_về> tên_phương_thức2(danh_sách_tham_số)
  {
```

```
    // phần thân phương thức
  }
  ...
  <access specifier> <kiểu_trả_về> tên_phương_thứcN(danh_sách_tham_số)
  {
    // phần thân phương thức
  }
}
```

- Access specifier xác định các qui tắc truy cập cho các thành viên cũng như chính lớp đó. Nếu không được đề cập, thì Access Specifier mặc định cho một kiểu lớp là **internal**. Chế độ truy cập mặc định cho các thành viên là **private**.
- kiểu_dữ_liệu xác định kiểu biến, và trả về kiểu dữ liệu mà phương thức trả về.

Để truy cập các thành viên lớp, bạn sử dụng toán tử dot (.). Toán tử dot (.) liên kết với tên của một đối tượng với tên của một thành viên.

1.3.8 Kế thừa

Một trong những khái niệm quan trọng nhất trong lập trình hướng đối tượng là **Tính kế thừa (Inheritance)**. Tính kế thừa cho phép chúng ta định nghĩa một lớp trong điều kiện một lớp khác, mà làm cho nó dễ dàng hơn để tạo và duy trì một ứng dụng. Điều này cũng cung cấp một cơ hội để tái sử dụng tính năng code và thời gian thực thi nhanh hơn.

Khi tạo một lớp, thay vì viết toàn bộ các thành viên dữ liệu và các hàm thành viên mới, lập trình viên có thể nên kế thừa các thành viên của một lớp đang tồn tại. Lớp đang tồn tại này được gọi là **Base Class - lớp cơ sở**, và lớp mới được xem như là **Derived Class – lớp thừa kế**.

Một lớp có thể được kế thừa từ hơn một lớp khác, nghĩa là, nó có thể kế thừa dữ liệu và hàm từ nhiều Lớp hoặc Interface cơ sở.

Cú pháp để tạo lớp kế thừa trong C# là:

```
<acess-specifier> class <base_class>
{
  ...
}
class <derived_class> : <base_class>
{
  ...
}
```

Đa kế thừa trong C#

C# không hỗ trợ đa kế thừa. Tuy nhiên, bạn có thể sử dụng Interface để triển khai đa kế thừa. Một Interface được định nghĩa như là một giao ước có tính chất cú pháp (syntactical contract) mà tất cả lớp kế thừa Interface đó nên theo. Các Interface được khai báo bởi sử dụng từ khóa interface trong C#. Nó tương tự như khai báo lớp. Theo mặc định, các lệnh Interface là puclic. Ví dụ sau minh họa một khai báo Interface trong C#:

```
public interface ITransactions
{
  // các thành viên của interface
  //các phương thức
  void hienThiThongTinGiaoDich();
  double laySoLuong();
}
```

1.3.9 Sự kiện (Event)

Sự kiện (Event) là các hành động của người dùng, ví dụ như nhấn phím, click, di chuyển chuột, … Các Application cần phản hồi các sự kiện này khi chúng xuất hiện. Ví dụ, các ngắt (interrupt). Các sự kiện (Event) được sử dụng để giao tiếp bên trong tiến trình.

Sử dụng Delegate với Event trong C#

Các Event được khai báo và được tạo trong một lớp và được liên kết với Event Handler bởi sử dụng các Delegate bên trong cùng lớp đó hoặc một số lớp khác. Lớp mà chứa Event được sử dụng để công bố event đó. Điều này được gọi là lớp Publisher. Một số lớp khác mà chấp nhận Event này được gọi là lớp Subscriber. Các Event trong C# sử dụng mô hình Publisher-Subscriber.

Một Publisher trong C# là một đối tượng mà chứa định nghĩa của event và delegate đó. Mối liên hệ event-delegate cũng được định nghĩa trong đối tượng này. Một đối tượng lớp Publisher triệu hồi Event và nó được thông báo tới các đối tượng khác.

Một Subscriber trong C# là một đối tượng mà chấp nhận event và cung cấp một Event Handler. Delegate trong lớp Publisher triệu hồi phương thức (Event Handler) của lớp Subscriber.

Khai báo Event trong C#

Để khai báo một Event bên trong một lớp, đầu tiên một kiểu delegate cho Event đó phải được khai báo. Ví dụ:

```
public delegate void BoilerLogHandler(string status);
```

Tiếp theo, chính Event đó được khai báo, bởi sử dụng từ khóa event trong C#:

```
public event BoilerLogHandler BoilerEventLog;
```

Code trên định nghĩa một delegate với tên là *BoilerLogHandler* và một Event với tên là *BoilerEventLog*, mà triệu hồi delegate đó khi nó được tạo ra.

Event trong thư viện .NET

Các Event ví dụ như KeyDown, GotFocus, Load của Form, Application.ApplicationExit, Application.Idle ... đều xây dựng từ một delegate là EventHandler

```
public delegate void EventHandler(object sender, EventArgs e);
public delegate void EventHandler(object sender, TEventArgs e)
    where TEventArgs : EventArgs;
```

Như vậy bạn có thể sử dụng luôn delegate EventHandler để xây dựng các Event của riêng mình sử dụng cho các Publisher, chỉ cần xây dựng các lớp phái sinh từ EventArgs với mục đích thêm vào các tham số riêng khi gửi sử kiện. Ví dụ:

```
public class MyEventArgs : EventArgs
{
   public MyEventArgs(string data)
   {
      this.data = data;
   }

   private string data;

   public string Data {
      get { return data; }
   }
}
```

2 Tìm hiểu Winform

2.1 Tổng quan

Winform (tên đầy đủ là Windows-form) là một công cụ dùng để tạo ứng dụng chạy trên nền tảng hệ điều hành Windows. Chúng ta có thể tạo ra giao diện của một ứng dụng bằng code hoặc sử dụng kéo thả thông qua phần mềm lập trình Visual Studio. Điều đó sẽ giúp người lập trình nhanh chóng tạo được giao diện ưng ý theo đúng bản vẽ đề ra, và cũng để khách hàng có thể xem trước được giao diện ứng dụng. Winform có những điểm mạnh của ngôn ngữ Visual Basic : dễ sử dụng, hỗ trợ mô hình RAD đồng thời kết hợp với tính linh động, hướng đối tượng của ngôn ngữ C#. Việc tạo ứng dụng Windows trở lên hấp dẫn và quen thuộc với các lập trình viên vì nó có tính ứng dụng cao.

2.2 Cơ bản về Winform

2.2.1 Form

Form là cửa sổ được hiện lên ngay khi mở ứng dụng, chứa tất cả các phần tử con như button, textbox,... Để lựa chọn những phần tử con dùng để tạo nên các chức năng của phần mềm và thêm vào form, hãy nhấn vào **Toolbox**. Để thay đổi các đặc điểm, tạo event,... cho mỗi phần tử, lựa chọn **Properties**.

Một số cài đặt về form:

- Vị trí ban đầu của form:
  Với 1 ứng dụng, ta có thể muốn form hiển thị ngay giữa màn hình, hoặc ta có thể muốn form này hiển thị giữa form kia,...Lựa chọn các thông số trong **StartPosition** để cài đặt theo ý muốn

Trị số Vị trí khởi đầu	Kết quả
Manual	Hiển thị form ở vị trí theo giá trị của **property Location** của form
CenterScreen	Hiển thị form ở ngay giữa màn ảnh
CenterParent	Hiển thị form ở ngay giữa form chủ (owner) của nó
WindowsDefaultLocation	Hiển thị form ở vị trí default của cửa sổ
WindowsDefaultBounds	Hiển thị form ở vị trí default của cửa sổ, với kích thước default của cửa sổ

- Đường viền của form:
  Một form thường có các đường viền (border) của nó, ta có thể thay đổi giá trị đường viền để form có những hình ảnh hiển thị khác nhau. Bạn có thể thay đổi các giá trị trong **FormBorderStyle**

Loại	Giá trị	Diễn giải
None	0	Không có đường biên
FixedSignle	1	Tương tự như FixedDialog
Gfixed3D	2	Trông giống như 3 chiều
FixedDialog	3	Như hộp hội thoại
Sizable	4	Mặc nhiên
FixedToolWindow	5	Thanh caption nhỏ và không có nút close
SizableToolWindow	6	FizedToolWindow nhưng đường biên mỏng.

- Một số thuộc tính khác:

Loại	Giá trị	Diễn giải
ControlBox	True False	Có hay không có 3 nút min, max, và close trên góc phải
Maximinzebox	True False	Có hay không có nút max
Miniminzebox	True False	Có hay không có nút min
TopMost	True False	Form này có luôn nằm lên trên hết các form khác hay không
Caption	Text	Đoạn text nằm trên thanh bar trên đầu của form
Name	Text	Tên của form.

Có rất nhiều thuộc tính khác nhau có thể tùy chọn thay đổi, mỗi phần tử khác nhau sẽ có một số thuộc tính khác nhau.

- Sự kiện của form:
  Trong lập trình Visual, điều quan trọng nhất là xử lý các sự kiện. Khi lập trình, thường ta chỉ thực hiện các thao tác kéo thả là ta có thể tạo được một giao diện hoàn chỉnh. Tuy nhiên, để giao diện đó hoạt động được theo đúng yêu cầu của ta thì ta buộc phải lập trình cho các sự kiện của từng hay nhiều control trên form đó. Sau đây là một số sự kiện của form
  - Sự kiện Load:

```
private void Form3_Load(object sender, System.EventArgs e)
{
//Code xử lý cho quá trình form được load lên.
}
```

  - Sự kiện Closed:

```
private void Form3_Closed(object sender, System.EventArgs e)
{
//Code xử lý cho quá trình form đang được đóng lại.
}
```

  - Sự kiện Click (khi ta click mouse lên trên form):

```
private void Form3_Click(object sender, System.EventArgs e)
{
//Code xử lý khi chuột được click lên trên form.
}
```

Để tạo 1 hàm xử lý sự kiện (event function), cách tốt nhất là bạn bấm double click chuột lên đối tượng mà bạn muốn tạo hàm để xử lý sự kiện đó. Tuy nhiên, cách này thì VS .Net nó chỉ phát sinh 1 hàm xử lý sự kiện mặc định của nó (với form thi hàm mặc định xử lý sự kiện là Form_Load. Do vậy, bạn hãy mở cửa sổ **Properties**

của đối tượng đó, chọn tab các events, nó sẽ liệt kê các events mà đối tượng, sau đó bạn double click vào sự kiện mà bạn muốn, và một hàm xử lý sự kiện được tạo ở bên phần source code và bạn chỉ việc viết thêm code để xử lý sự kiện này mà không cần quan tâm đến bằng cách nào mà hệ thống kiểm tra và xử lý nó.

2.2.2 Button

Button control là một đối tượng nút nhấn được các nhà phát triển ngôn ngữ lập trình sẵn và để trong thư viện của Visual Studio .Net. Button control cho phép người dùng click chuột vào nó và nó sẽ thực thi một hay nhiều đoạn mã nào đó mà do người lập trình chỉ định thông qua các events mà nó nhận được. Khi một button được click thì nó sẽ sinh ra một số các sự kiện (events) như Mouse Enter, MouseDown, MouseUp, v.v… và tuỳ với các sự kiện này mà chúng ta có thể lựa viết các đoạn code để xử lý cho phù hợp.

- Một số thuộc tính của button:

Tên Thuộc tính	**Giá Trị**	**Diễn giải**
Name	Text	Tên cho button
Text	Text	Chữ sẽ hiển thị trên button
Font	Hộp hội thoại font	Chọn font chữ cho text (chữ hiển thị trên button)
BackColor	Hộp màu	Chọn màu nền cho button
ForeColor	Hộp màu	Chọn màu chữ cho button.
TextAlign	TopLeft – TopCenter – TopRight	Canh lề chữ: trên bên trái – giữa phải.
	MiddleLeft – MiddleCenter – MiddleRight	ở giữa – bên trái – bên phải
	BottomLeft – BottomCenter – BottomRight	bên dưới trái – giữa – phải

FlatStyle	Flat	Button sẽ bằng phẳng
	Popup	Giống Flat nhưng khi move mouse lên trên nó thì giống standard và khi bỏ mouse ra thì nó trở lại như cũ (Flat)
	Standard	Mặc định của hệ thống
	System	Giống standard nhưng sẽ thay đổi theo hệ điều hành
BackGroundImage	đường dẫn file hình	Hình bạn chỉ định sẽ làm nền cho button. Nếu hình không đủ lớn bằng button thì hình sẽ lặp lại (giống như bạn chọn chế độ Title khi bạn set background cho màn hình window).
Image	đường dẫn file hình	Hình sẽ làm nền cho button (giống như bạn chọn Center khi set hình background cho window).

- Thuộc tính Anchor:
  Anchor (bỏ neo) là thuộc tính rất tiện dụng mà Visual Studio .Net mới đưa vào. Nó cho phép ta định vị trí của một control trên form khi ta resize một form. Khi con tàu bỏ neo là nó đỗ ở đó. Dù con nước chảy thế nào, con tàu vẫn nằm yên một chỗ vì nó đã được cột vào cái neo. Control trong .NET có property **Anchor** để ta chỉ định nó được cố định vào góc nào của form: **Left, Right, Bottom** hay **Top**.
- Các sự kiện của button:
  Trong các sự kiện của button thì chỉ có sự kiện Click chuột là quan trong nhất. Sự kiện click chuột là sự kiện mặc định của control button, do đó, bạn chỉ cần double click chuột vào button cần tạo sự kiện là VS .Net sẽ mở cửa sổ source code ra và tự động generate một hàm xử lý sự kiện click chuột cho bạn.
  private void button1_Click(object sender, System.EventArgs e)
  {
  //Bạn sẽ đánh code cho phần xử lý sự kiện ở đây.

}

với **button1_Click** thì **button1** chính là tên của control button mà bạn tạo sự kiện Click chuột cho nó.

2.2.3 Label

Label (nhãn) là thành phần đơn giản nhất và cũng là một trong những thành phần quan trọng nhất trong lập trình visual. Đối tượng nhãn chỉ để dùng trình bày một chuỗi văn bản thông thường nhằm mục đính mô tả thêm thông tin cho các đối tượng khác. Ta cũng có thể dùng nhãn để làm công cụ đưa kết quả ra màn hình dưới dạng một chuỗi.

- Một số thuộc tính của label:

Tên Thuộc tính	Giá Trị	Diễn giải
BorderStyle	None (mặc định)	Không có đường viền
	FixedSingle	Có đường viền bao quanh
	Fixed3D	3D, trông nó giống như bị lõm xuống (giống textbox)

Đối với nhãn, ta cũng có thể sử dụng anchoring để chỉ định vị trí nhãn trên form khi ta resize form.

Trong lập trình Visual, mọi đối tượng đều có sự kiện cho riêng nó, tuy nhiên đối với nhãn, ta thường không sử dụng sự kiện vì nhãn chỉ có chức năng thông dụng là hiển thị một câu thông báo hay kết quả ra màn hình.

2.2.4 Textbox

Textbox (ô nhập) là đối tượng để nhập văn bản vào. Đây cũng là một trong những control thông dụng nhất trong lập trình visual.

- Một số thuộc tính của textbox:

Tên Thuộc tính	Giá Trị	Diễn giải
MaxLength	integer	Chiều dài tối đa dữ liệu nhập (ký tự)
Multiline	True False	Cho hay không cho textbox cao lên để cho phép nhập nhiều dòng.
PasswordChar	character	Nếu bạn muốn textbox này là 1 password field thì bạn chỉ cần chỉ định character mà bạn muốn hiển thị khi bạn nhập password vào thuộc tính này.
ReadOnly	True False	Cho hay không cho phép nhập liệu vào textbox này.
RightToLeft	Yes No	Yes – Nếu bạn muốn canh lề phải dữ liệu nhập.

Thuộc tính ScrollBar chỉ có tác dụng khi chọn Multiline = True. Đây là thuộc tính giúp bạn điều chỉnh thanh cuộn: None-Không có thanh cuộn;

Horizontal-Chỉ có thanh cuộn đứng; Vertical-Chỉ có thanh cuộn ngang; Both-Có cả 2 thanh cuộn đứng và ngang.

Cũng như label, Textbox thường chức năng chính là để nhập liệu nên nó cũng ít khi phải sử dụng các sự kiện.

2.2.5 Checkbox

Checkbox (ô đánh dấu) tương tự như nút nhấn, chỉ khác là chúng dùng để biểu hiện trạng thái chuyển đổi giữa có/không (yes/no) hoặc tắt mở (on/off). Mỗi lần ta nhấp chuột vào ô chọn hay dùng nhấn phím space bar khi checkbox đang focus thì nó thay đổi trạng thái từ đươc đánh dấu sang bỏ đánh dấu hoặc ngược lại.

- Một số thuộc tính của checkbox:

Tên Thuộc tính	Giá Trị	Diễn giải
CheckAlign	TopLeft – TopCenter – TopRight	**Canh lề checkbox**: trên bên trái – giữa phải của khung checkbox
	MiddleLeft – MiddleCenter – MiddleRight (mặc định)	ở giữa – bên trái – bên phải của khung checkbox
	BottomLeft – BottomCenter – BottomRight	bên dưới trái – giữa – phải của khung checkbox
RightToLeft	Yes No	Chữ nằm bên phải hay bên trái ô checkbox.
FlatStyle	Flat	checkbox sẽ bằng phẳng
	Popup	Giống Flat nhưng khi move mouse lên trên nó thì giống standard và khi bỏ mouse ra thì nó trở lại như cũ (Flat)
	Standard (mặc định)	checkbox lõm xuống (3D)
	System	Giống standard nhưng sẽ thay đổi theo hệ điều hành
Checked	True False	Chọn checkbox (đánh dấu) hay không chọn checkbox.
CheckState	checked	checkbox đang được đán dấu
	unchecked	checkbox không được đánh dấu
	indeterminate trạng thái này chỉ có tác dụng khi thuộc tính ThreeState = true	checkbox ở trạng thái không hoạt động
ThreeState	true false	Cho phép checkbox có 3 trạng thái: chọn – không chọn – không tác dụng

- Các sự kiện của checkbox:
  Checkbox cũng có nhiều sự kiện nhưng ta chỉ xét một sự kiện quan trọng nhất và thông dụng nhất ở đây là sự kiện Click. Sự kiện click của checkbox cũng giống như sự kiện click của button, tuy nhiên, ta cần phải kiểm tra để biết được là khi ta click chuột này thì trạng thái chọn (checked) hay là trạng thái bỏ chọn (unchecked).
  Để kiểm tra xem là checkbox đang ở trạng thái nào, bạn có 2 cách kiểm tra:
  - Dùng thuộc tính **checked** của checkbox: có 2 trạng thái
  - Dùng thuộc tính **checkstatus** của checkbox: có thể có 3 trạng thái

2.2.6 PictureBox

PictureBox là một khung để hiển thị hình ảnh (bitmap, GIF, icon, JPEG) bên trong khung đó. Khi lập trình phần mềm cần phải sử dụng tới hình ảnh thì chúng ta không thể bỏ qua Control này. Bạn có thể thiết lập các thuộc tính của ảnh khi thiết kế Form hoặc khi chạy chương trình.
Thuộc tính **SizeMode** dùng để thay đổi cách hiển thị ảnh. Những cách hiển thị hình ảnh trên PictureBox:

- AutoSize: Tự động điều chỉnh kích cỡ ảnh.
- CenterImage: Căn giữa ảnh.
- Normal: Đặt góc trái phía trên của ảnh vào vị trí góc trái phía trên của PictureBox.
- StretchImage: Giãn nở kích thước theo PictureBox.

2.2.7 MessageBox

MessageBox là một lớp (class) nằm trong System.Windows.Forms có một phương thức **Show** để hiển thị thông báo. MessageBox không cài đặt trong form mà sẽ hiện lại trong 1 cửa sổ riêng. Có rất nhiều kiểu thông báo, bạn có thể điều chỉnh nội dung thông báo, tiêu đề, các nút OK-Cancel, biểu tượng,...
Sau đây là những kiểu MessageBox khác nhau trong Winform:

- Kiểu đơn giản:

MessageBox.Show("Xin chào!");

Đây là kiểu thông báo đơn giản nhất, chỉ có nội dung và nút OK, chưa bao gồm biểu tượng, tiêu đề,...

- Kiểu có tiêu đề:

MessageBox.Show("Xin chào!","Thông báo");

MessageBox sẽ có thêm tiêu đề là “Thông báo”.

- Cài đặt thêm các loại nút nhấn:

Để cài đặt nút bấm, ta cũng thêm 1 tham số kiểu enum là **MessageBoxButtons.<loại nút>**. Các loại nút có sẵn bao gồm **AbortRetryIgnore**,**OK**,**OKCancel**,**RetryCancel**,**YesNo**,**YesNoCancel**. Ví dụ:

MessageBox.Show("Xin chào! Tôi là C#", "Thông báo", MessageBoxButtons.AbortRetryIgnore);

- Cài đặt thêm Icon:

Để thêm vào icon ta thêm tham số kiểu enum là **MessageBoxIcon.<loại icon>**, có nhiều loại nhưng phổ biến là **Warning** (tam giác vàng có dấu chấm than), **Error** (hình tròn đỏ có chữ X), **Information** (hình tròn xanh lam có chữ i), **Question** (hình tròn lam có dấu chấm hỏi). Ví dụ:

MessageBox.Show("Xin chào! Tôi là C#", "Thông báo", MessageBoxButtons.OKCancel, MessageBoxIcon.Question);

Trong MessageBox cũng cần xử lí các sự kiện, sự kiện chính là khi click vào một button nào đó trên MessageBox. Sau đây là một ví dụ cụ thể:

Giả sử bạn có một Form, và bạn muốn khi người dùng nhấp vào nút Close trên thanh tiêu đề thì sẽ có thông báo hỏi người dùng có muốn thoát chương trình. Nếu người dùng chọn Yes, chương trình sẽ kết thúc, chọn No sẽ không tắt chương trình.

Ta xử lý sự kiện FormClosing, tức là một Form đang đóng lại (nháy vào nút X hoặc lệnh this.Close(),...). Bạn sẽ dùng một biến kiểu DialogResult để lưu lại kết quả trả về của phương thức MessageBox.Show()

DialogResult dlr = MessageBox.Show("Bạn muốn thoát chương trình?", "Thông báo", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

Đến lúc này ta chỉ cần xét giá trị của biến *dlr* để rẽ nhánh thôi. Vì Form đang đóng nên nếu người dùng nhấn No thì sẽ hoãn hành động đóng lại, tham số sự kiện của *FormClosing* là **e** nên ta thực hiện như sau:

```
DialogResult dlr = MessageBox.Show("Bạn muốn thoát chương trình?",
    "Thông báo", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
if (dlr == DialogResult.No) e.Cancel = true;
```

3 Tạo kết nối LAN

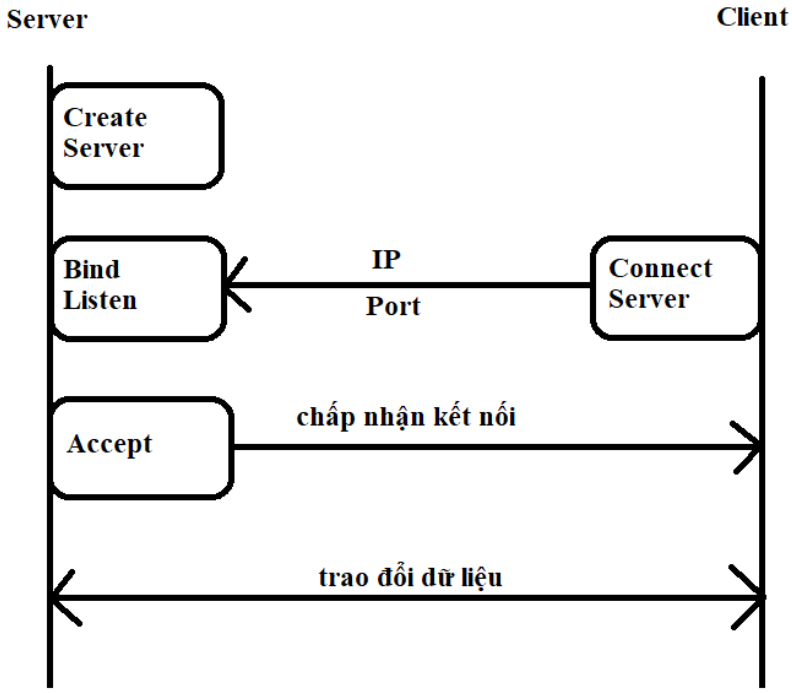
3.1 Socket

Kết nối mạng LAN sử dụng giao thức TCP/IP thông qua Socket. Socket là một điểm cuối (end-point) của liên kết truyền thông hai chiều (two-way communication) giữa hai chương trình chạy trên mạng. Các lớp Socket được sử dụng để biểu diễn kết nối giữa client và server, được ràng buộc với một cổng port

(thể hiện là một con số cụ thể) để các tầng TCP (TCP Layer) có thể định danh ứng dụng mà dữ liệu sẽ được gửi tới.
Lập trình socket là lập trình cho phép người dùng kết nối các máy tính truyền tải và nhận dữ liệu từ máy tính thông qua mạng. Hiểu đơn giản, socket là thiết bị truyền thông hai chiều gửi và nhận dữ liệu từ máy khác.
Là một giao diện lập trình ứng dụng mạng, socket giúp các bạn lập trình kết nối các ứng dụng để truyền và nhận giữ liệu trong môi trường có kết nối Internet bằng cách sử dụng phương thức TCP/IP và UDP.
Khi cần trao đổi dữ liệu cho nhau thì 2 ứng dụng cần phải biết thông tin tối thiểu là IP và số hiệu cổng của ứng dụng kia.
Game cờ caro sử dụng Stream Socket (socket hướng kết nối) thích hợp với mô hình Server-Client. Server tạo kết nối, lắng nghe và chấp nhận từ Client. Client cần kết nối tới Server.

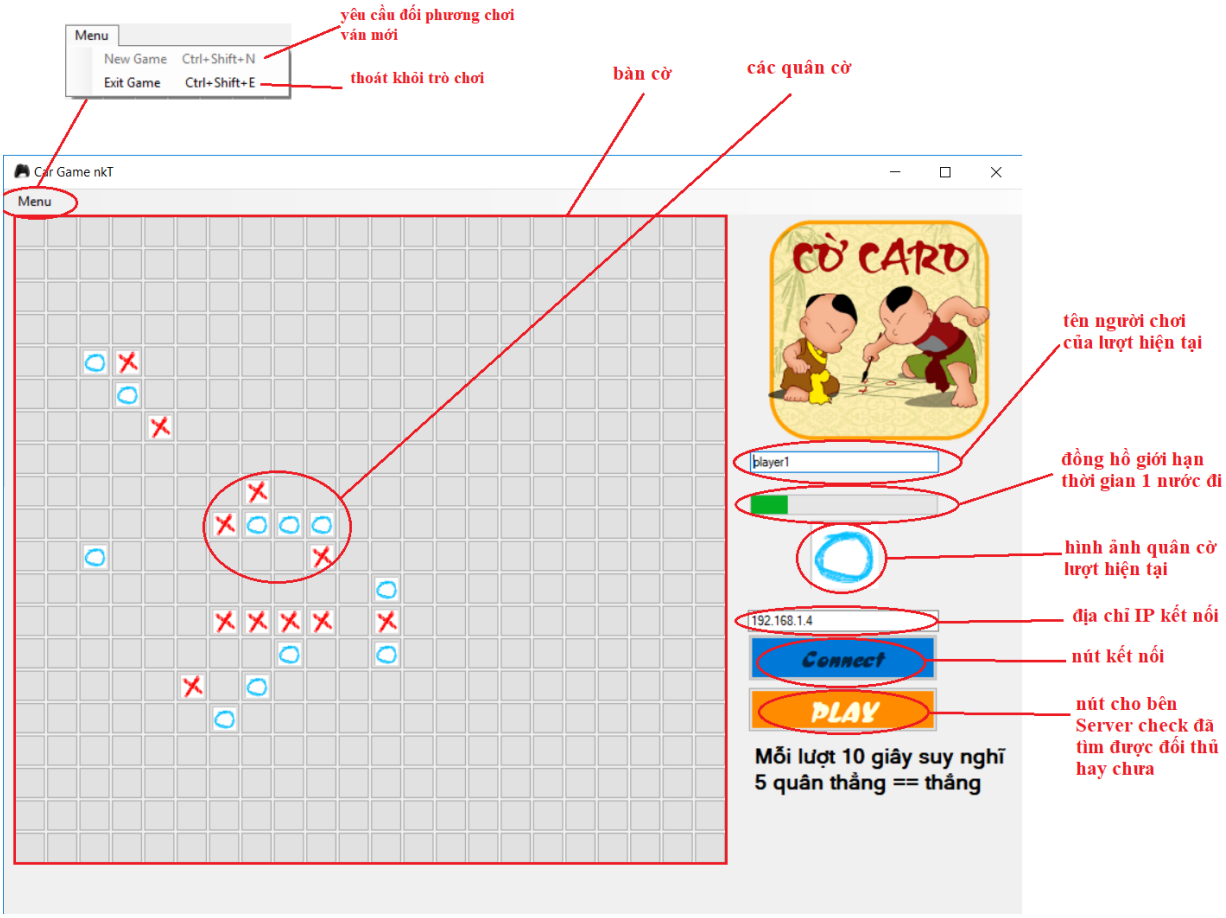
3.2 Server và Client

Khi mới bắt đầu, tạo một Socket. Nếu chưa có kết nối sẵn, nó trở thành Server để tạo kết nối. Nếu đã có tín hiệu kết nối, nó đồng ý kết nối và trở thành Client.
Server khởi tạo một kết nối tới một địa chỉ IP và một cổng xác định. Sau đó Server sẽ ngồi lắng nghe tín hiệu từ Client thông qua phương thức Bind và Listen.
Client kết nối với Server cũng qua một địa chỉ IP và cổng xác định. Nếu đồng ý kết nối, nó sẽ gửi tín hiệu để Server nhận biết. Lúc này Server sẽ Accept, 2 bên đã thông nhau, có thể truyền nhận dữ liệu.



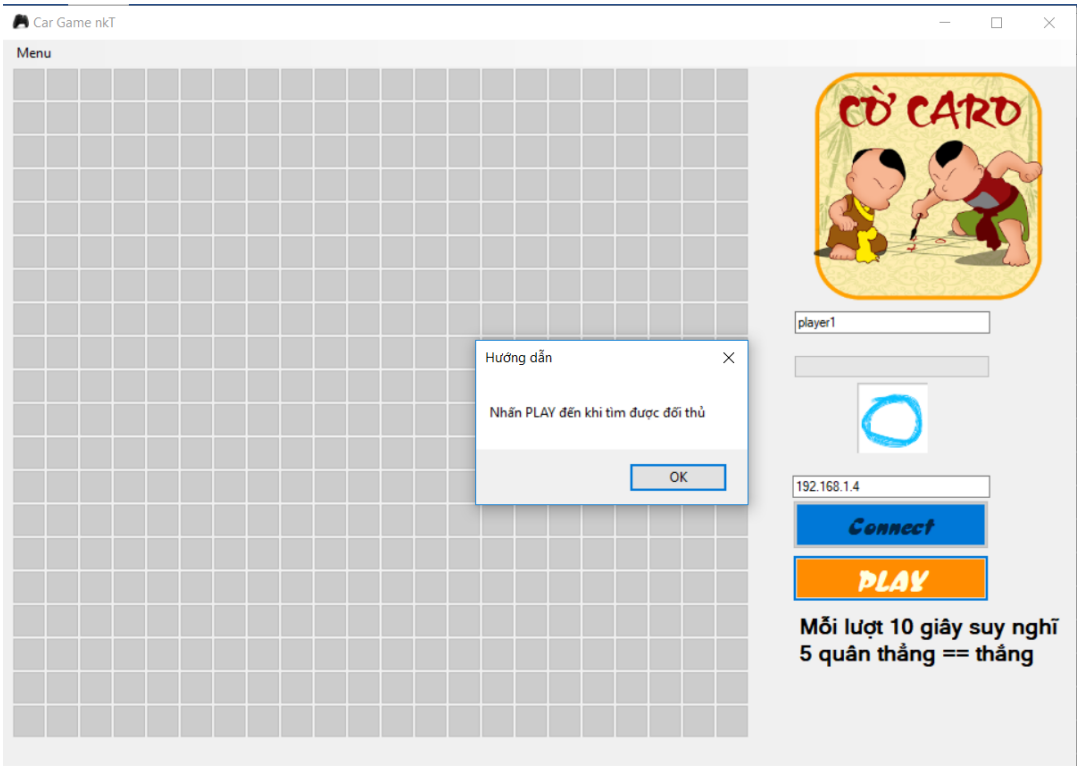
4 Game cờ Caro kết nối mạng LAN

4.1 Giao diện

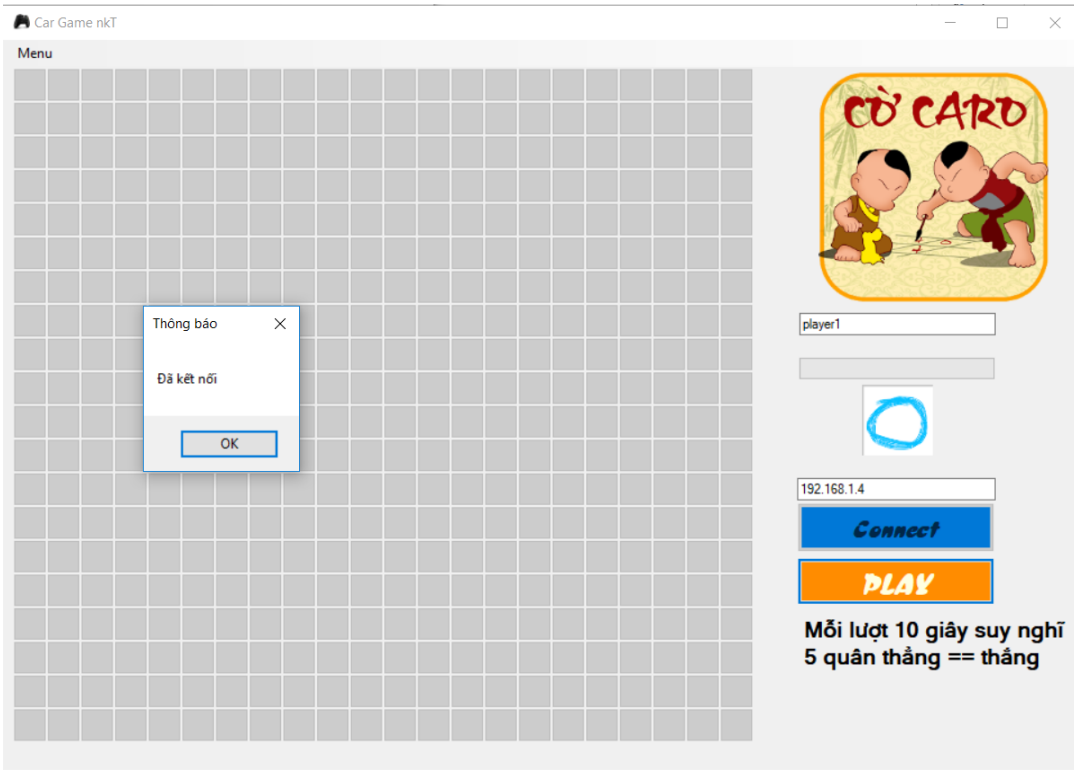


4.2 Cách kết nối giữa Server và Client

Khi khởi động, nhấn Connect. Nếu là Server sẽ hiện ra MessageBox hướng dẫn. Nếu là Client thì sẽ hiện ra MessageBox thông báo đã kết nối được.

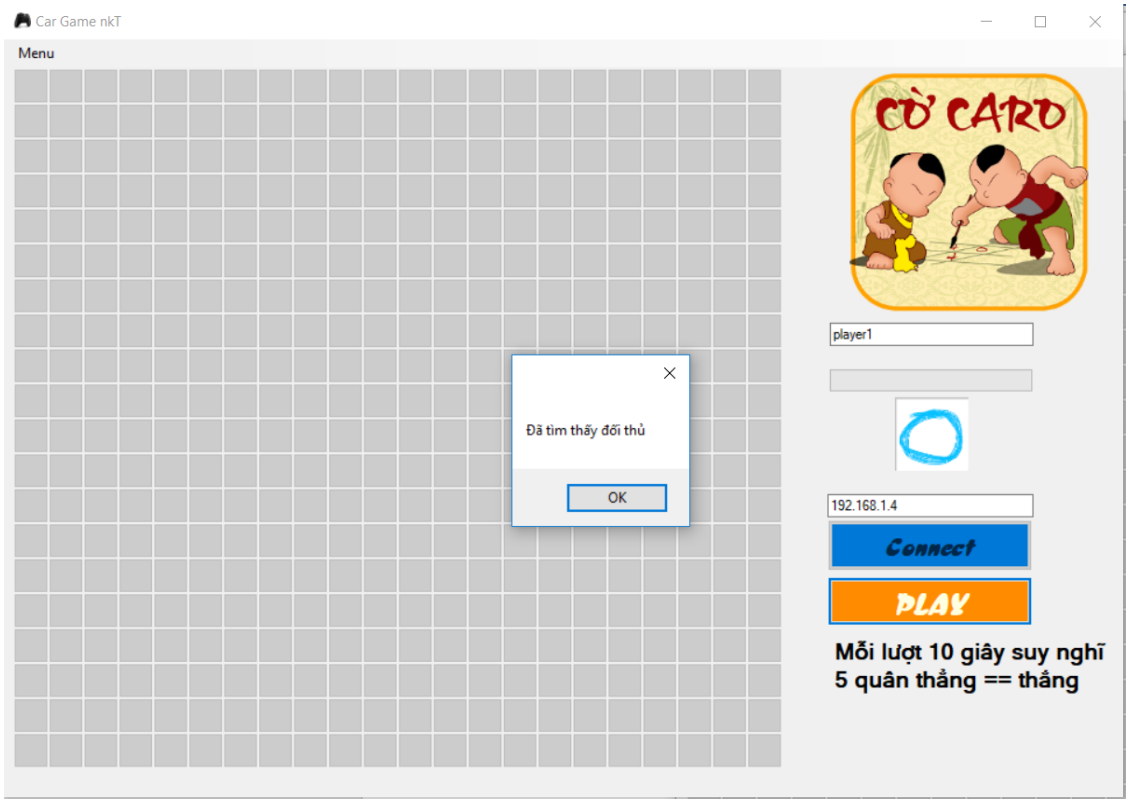


Server

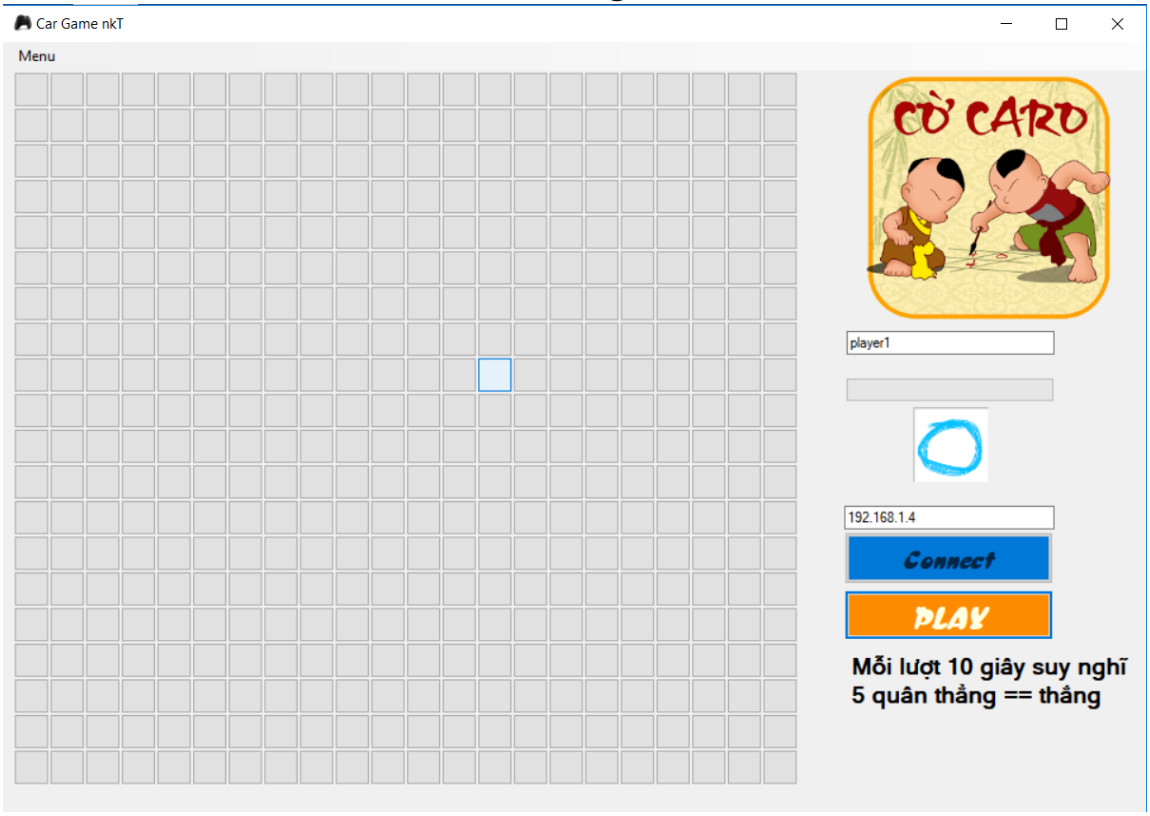


Client

Bên Server nhấn PLAY liên tục đến khi có thông báo sau hiện ra. Nhấn OK chương trình sẽ mở bàn cờ cho bên Server đi trước.

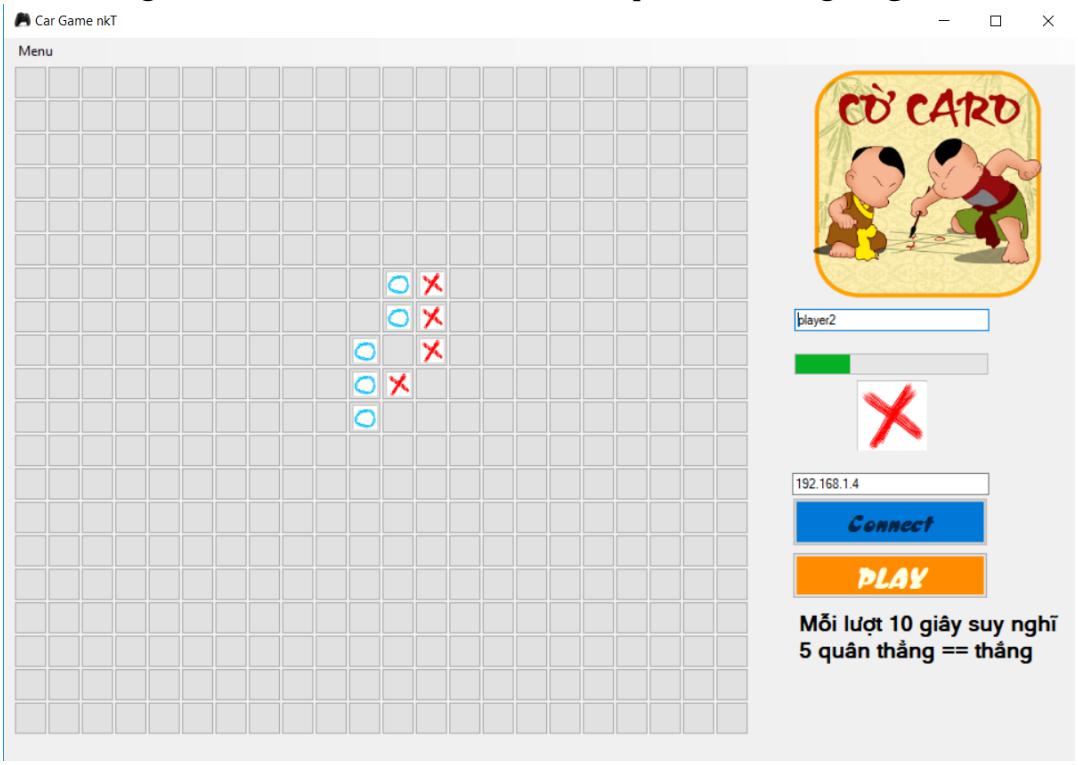


Hiện thông báo

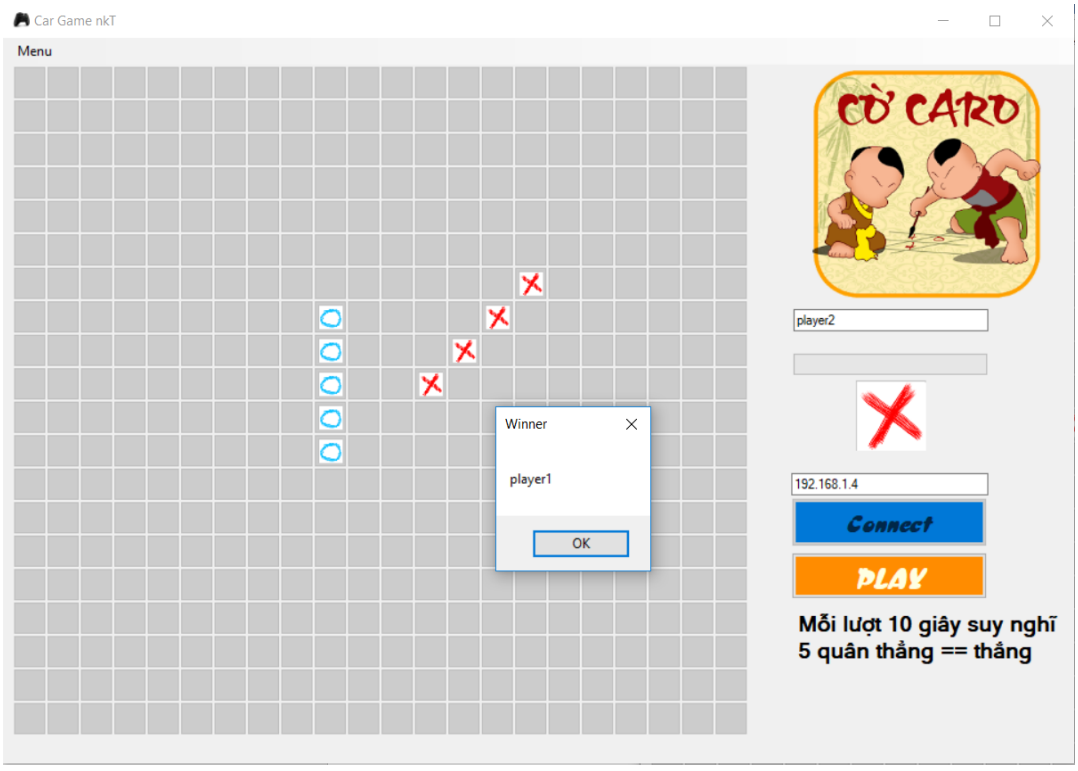


Nhấn OK, bàn cờ đã mở

Lúc này 2 bên đã kết nối với nhau, có thể đánh quân cờ qua lại. Khi nào đến lượt đánh thì chương trình sẽ mở bàn cờ và hiển thị quân cờ tương ứng với bên đó.

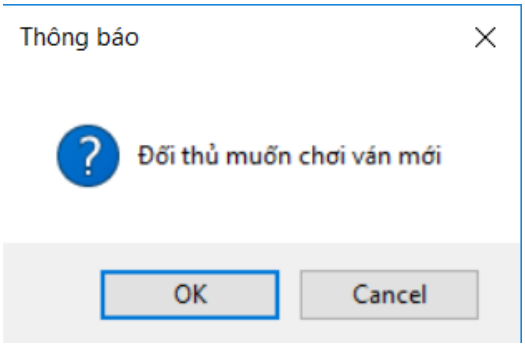


Khi có người chơi thắng, chương trình sẽ hiển thị thông báo bên thắng và khóa bàn cờ lại.

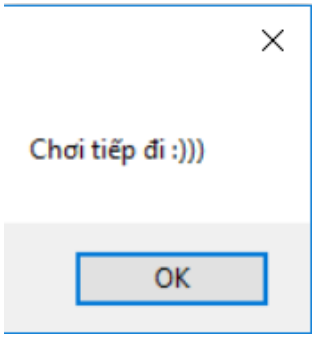


Nếu muốn chơi ván mới, nhấn Menu -> New Game hoặc sử dụng phím tắt Ctrl+Shift+N. Đối phương sẽ nhận được yêu cầu của bạn. Nhấn OK thì cả 2 cùng reset bàn cờ, chơi ván mới. Nhấn Cancel thì tiếp tục chơi, bên yêu cầu nhận được thông báo.

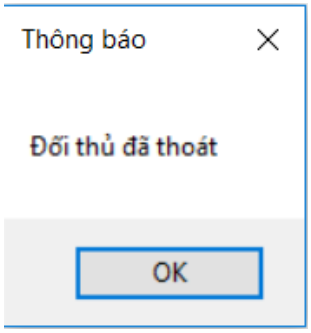
Yêu cầu gửi cho đối phương:



Thông báo khi không đồng ý ván mới:



Khi đối thủ thoát, sẽ có thông báo gửi về. Lúc này kết nối đã bị ngắt, chương trình như lúc mới khởi động.



4.3 Các lớp và một số hàm cơ bản trong lớp

4.3.1 Form1.cs [Design]

Lớp thiết kế giao diện bằng kéo thả và hiệu chỉnh các thông số.

4.3.2 Const.cs

Lớp lưu các hằng số cho chương trình: độ dài rộng bàn cờ; độ dài rộng quân cờ; thông số cho thanh đồng hồ.

4.3.3 Player.cs

Lớp có chức năng lưu tên và hình ảnh quân cờ của mỗi người chơi.

4.3.4 PlayInfo.cs

Lưu thông tin vị trí quân cờ và người chơi hiện tại.

4.3.5 CaroBoardManager.cs

Xử lí thông tin về bàn cờ, tên người chơi và hình ảnh quân cờ của người chơi thay đổi theo mỗi lượt.

- drawBoard():Hàm vẽ bàn cờ với số kích thước trong Const.cs
- btn_Click(object sender, EventArgs e): Hàm xử lí khi có sự kiện click vào ô trên bàn cờ. Khi click vào ô trống, hình ảnh quân cờ của người chơi sẽ điền vào đó, đổi lượt cho người chơi kia đồng thời sẽ xét xem đã có người thắng cuộc hay chưa.
- OtherPlayerMark(Point point): xác định vị trí quân cờ để truyền dữ liệu cho đối phương.
- EndGame(): xử lí sự kiện khi có quân cờ làm kết thúc trò chơi
- isEndGame(Button btn): xác định kết thúc trò chơi
  - isEndHor(Button btn): kết thúc khi 5 quân liền nhau theo chiều ngang
  - isEndVer(Button btn): kết thúc khi 5 quân liền nhau theo chiều dọc
  - isEndDia(Button btn): kết thúc khi 5 quân liền nhau theo đường chéo phải
  - isEndDiaSub(Button btn): kết thúc khi 5 quân liền nhau theo đường chéo trái
- getPoint(Button btn): lấy vị trí của quân cờ

4.3.6 SocketData.cs

Các thông tin của Socket khi được truyền:

- Command: lệnh truyền
- Point: vị trí quân cờ truyền
- Messege: lời nhắn

public enum SocketCommand chứa tên các lệnh cần gửi:

- SEND_POINT: truyền quân cờ
- NEW_GAME: ván đấu mới
- RQ_NEW_GAME: yêu cầu ván đấu mới
- QUIT: thoát trò chơi
- START: bắt đầu trò chơi

4.3.7 SocketManager.cs

Xử lí các hoạt động của Socket

- Client: xử lí socket nếu là Client

    - ConnectServer(): kết nối Server
- Server: xử lí socket nếu là Server
    - CreateServer(): tạo kết nối
- Both: xử lí chung cho cả Server và Client
    - Send(object data): nén dữ liệu thành mảng byte rồi gửi
    - Receive(): nhận dữ liệu kiểu byte, sau đó giải nén mảng đó ra thành object
    - CloseConnect(): ngắt kết nối cho Server
    - byte[] SerializeData(Object ob): Nén Object ob thành 1 mảng byte[]
    - object DeserializeData(byte[] byteArray): Giải nén mảng byte[] thành đối tượng Object
    - GetLocalIPv4(NetworkInterfaceType type): lấy IPv4 của card mạng đang dùng

4.3.8 Form1.cs

Lớp cài đặt giúp giao diện có thể hoạt động.

- CaroBoard_PlayerMarked(object sender, ButtonClickEvent e): Hàm xử lí khi có sự kiện 1 quân cờ được điền vào bàn cờ, nó sẽ gửi SEND_POINT, quân cờ vừa đánh rồi ngồi lắng nghe.
- CaroBoard_EndedGame(object sender, EventArgs e): xử lí sự kiện kết thúc trò chơi
- EndGame(): kết thúc trò chơi, khóa bàn cờ và thông báo người thắng
- NewGame(): tạo 1 ván đấu mới
- Form1_FormClosing(object sender, FormClosingEventArgs e): xử lí sự kiện khi thoát trò chơi. Khi người chơi ấn thoát sẽ có thông báo hỏi muốn thoát hay không. Nếu người chơi nhấn Cancel, trò chơi tiếp tục. Nếu người chơi nhấn OK sẽ gửi thông báo cho đối thủ, tắt chương trình
- buttonLAN_Click(object sender, EventArgs e): xử lí sự kiện khi nhấn vào nút “Connect” trong giao diện. Nếu là Server sẽ thực hiện tạo kết nối. Nếu là Client sẽ kết nối, hiện thông báo và truyền dữ liệu START để Server biết đã có người chơi.
- buttonPlay_Click(object sender, EventArgs e): xử lí sự kiện khi nhấn nút PLAY trên giao diện.
- Listen(): Hàm thực hiện chức năng lắng nghe
- ProcessData(SocketData data): Hàm cài đặt với mỗi dữ liệu gửi đi sẽ thực hiện những chức năng gì
    - NEW_GAME: tạo 1 ván đấu mới

- RQ_NEW_GAME: yêu cầu đối thủ chơi 1 ván đấu mới, sẽ truyền đi MessageBox thông báo. Nhấn OK thì cả 2 cùng reset bàn cờ, chơi ván mới. Nhấn Cancel thì tiếp tục chơi, bên yêu cầu nhận được thông báo.
- SEND_POINT: gửi đi quân cờ vừa đánh trên bàn cờ
- QUIT: khi thoát sẽ truyền thông báo cho đối phương biết mình đã thoát
- START: truyền dữ liệu cho Server biết đã có thể bắt đầu ván đấu

5 Kết luận

Sau thời gian tìm hiểu, học hỏi cùng với sự giúp đỡ của thầy giáo Thân Quang Khoát, em đã biết thêm một ngôn ngữ lập trình mới đó là C#, ứng dụng ngôn ngữ đó cùng với Winform để tạo nên một trò chơi kết nối mạng LAN. Phần mềm em tạo ra tuy đơn giản nhưng đã giúp em có thêm nhiều kiến thức về lập trình cũng như về mạng máy tính.

Tài liệu tham khảo

https://www.howkteam.vn/course/lap-trinh-game-caro-voi-c-winform-14
Tìm hiểu C#:
https://www.howkteam.vn/course/khoa-hoc-lap-trinh-c-can-ban-1
Tìm hiểu Winform:
http://info24h.vn/lap-trinh-xu-ly-giao-dien-trong-winform-voi-c-183.html
http://cunglaptrinh.blogspot.com/2015/03/lam-quen-voi-messagebox-trong-csharp.html
Tìm hiểu cách kết nối Client-Server:
https://sinhvientot.net/lap-trinh-mang-xay-dung-ung-dung-client-server-huong-ket-noi-tcp-socket/
https://yinyangit.wordpress.com/2011/06/22/socket-communication-with-tcp-client-server/comment-page-1/