

# Mục lục

## 1. Bài tập Design Pattern: Adapter Pattern

1.1. Tại sao phải có Adapter Pattern?

1.2. Adapter Pattern: Nội dung

1.3. Code minh họa

1.1.1. Không dùng adapter

1.1.2. Có adapter

## 2. Coding convention

2.1. Naming convention

2.2. Style convention

2.3. Solution convention

## 3. Bài tập cá nhân

3.1. Cơ sở kiến thức

3.1.1. Kiến thức về lập trình hướng đối tượng

3.1.2. Lý thuyết trò chơi: Giới thiệu về cờ tướng

3.2. Biểu đồ use-case

3.2.1. Biểu đồ use-case

3.2.2. Đặc tả biểu đồ use-case

3.3. Thiết kế chi tiết

3.3.1. Biểu đồ lớp (UML class diagram)

3.3.2. Lớp Game

3.3.2. Lớp XqBoard

3.3.3. Lớp trừu tượng Unit và các lớp con

3.4. Kết quả chương trình

Phụ lục: Tài liệu tham khảo

# 1. Bài tập Design Pattern: Adapter Pattern

## 1.1. Tại sao phải có adapter pattern?

Design pattern là các khuôn mẫu hoàn chỉnh được tối ưu cho kiến trúc lập trình hướng đối tượng. Nó sử dụng những tính chất cơ bản nhất của hướng đối tượng: tính kế thừa, tính đa hình, tính đóng gói,... để tạo nên các kiến trúc đáp ứng cho từng nhu cầu cụ thể.

Viết mã nguồn dựa trên design pattern giúp ta dễ dàng quản lý, bảo trì chương trình. Đa số các design pattern đề ra là nhằm mục đích đơn giản hóa client code hết mức có thể.

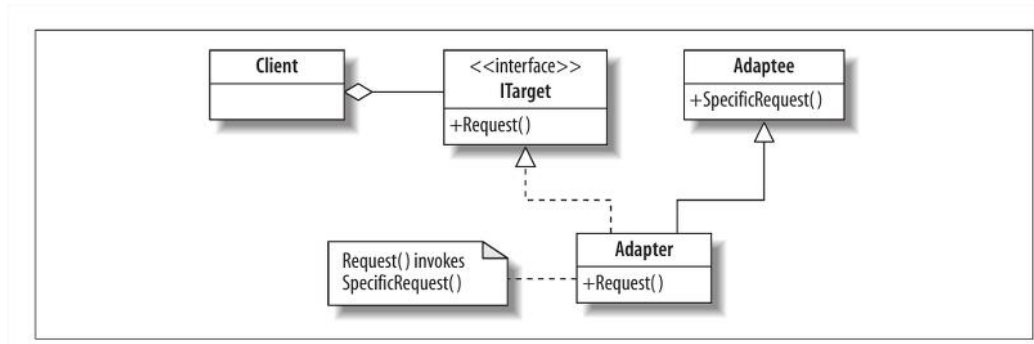
Adapter pattern là một trong số những design pattern quan trọng nhất, giúp cho một solution có thể phục vụ cho nhiều interface khác nhau mà không cần viết lại code.

Ví dụ, chúng ta muốn thiết kế chương trình quản lý một chiếc điện thoại di động. Khoảng 10 năm trước, một chiếc điện thoại di động chỉ được dùng để nghe, gọi, nhắn tin. Thế nên một giải pháp là chúng ta đi theo người sử dụng, thiết kế các hàm nghe, gọi, nhắn tin tương ứng. Thế nhưng theo thời gian, nhu cầu của người dùng ngày càng nâng cao, một chiếc điện thoại ngày nay phải nghe nhạc được, chơi game được, lướt Facebook được. Nếu theo giải pháp trên, bây giờ chúng ta lại viết thêm các hàm nghe nhạc, chơi game, lướt Facebook lại từ đầu thì tốn rất nhiều thời gian và công sức. Chưa kể code sẽ rất dài (vì nhu cầu của người dùng là muôn hình vạn trạng), không thống nhất, khó bảo trì.

Một giải pháp khác là khi thiết kế chương trình quản lý, ta chỉ cần thiết kế những hàm cơ bản nhất mà ứng dụng nào cũng cần như: đóng mở file, đóng mở ứng dụng, bật tắt máy, bật tắt loa, bật tắt GPS. Sau đó tùy vào nhu cầu của người dùng mà ta viết các hàm chuyển đổi tương ứng: ví dụ muốn nghe nhạc thì gọi hàm mở ứng dụng máy nghe nhạc → mở file nhạc cần nghe → bật loa. Khi này mã nguồn của chúng ta không những sẽ mềm dẻo, dễ dàng thích ứng với nhu cầu của người dùng mà lại còn trong sáng, dễ bảo trì.

Chiến lược kể trên là một minh họa cho việc sử dụng adapter pattern.

## 1.2. Adapter Pattern – Nội dung



Giả sử có một **ITarget** là interface bao gồm các hàm cần phải thi hành mà một trong số đó là hàm Request(), **Adaptee** là class chứa các thực thi có sẵn. Như sơ đồ trên, ta sẽ tạo một class mới: **Adapter** kế thừa cả interface **ITarget** lẫn class **Adaptee**. Request() trong **Adapter** sẽ thực hiện bằng cách gọi các hàm dạng SpecificRequest() kế thừa từ **Adaptee**.

Bằng cách này, **Adapter** mới ra đời không mất nhiều công sức và dòng lệnh (vì dựa trên hàm trong **Adaptee**) mà vẫn thi hành đầy đủ interface **ITarget**.

Trong thực tế, adapter pattern được sử dụng rất nhiều. Một ví dụ mà em biết liên quan đến game: Riot Games và Valve là hai nhà phát triển game nổi tiếng, mỗi thằng lại có một API riêng cho phép người phát triển lấy các thông tin về thông số kỹ thuật của game cũng như dữ liệu công khai của người chơi. Chẳng hạn em muốn viết một chương trình thu thập dữ liệu của người chơi và đưa ra bảng xếp hạng những người nào có thời gian chơi nhiều nhất. Thế thì giải pháp của em là đầu tiên xây dựng một interface **ITarget** bao gồm những hàm em mong muốn nhận được để có thể triển khai thuật toán sắp xếp và tính thời gian chơi trên đó. API của Riot Games và Valve bây giờ đóng vai trò là hai **Adaptee**. Em sẽ viết adapter cho từng loại cho phù hợp với **ITarget** mà em mong muốn.

Có nhiều loại adapter khác nhau. Ví dụ:

- Adapter 2 chiều (Two-way adapter): Adapter có cả những thực hiện của **ITarget** lẫn của **Adaptee** ban đầu, phù hợp sử dụng cho cả nhà sản xuất và người dùng cuối.
- Adapter đa hình (Pluggable adapter): 1 Adapter có thể dùng cho nhiều Adaptee khác nhau. Người dùng sau có quyền chọn Adaptee cần sử dụng.

### 1.3. Code minh họa

Smartphone as an **Adaptee**

```

class Smartphone
{
    protected int status = 0;
    //Off = 0, On = 1
    protected int isOpenAFile;
    public void turnOn(){
        status = 1;
        Console.WriteLine("Điện thoại đã được bật");
    }
    public void turnOff(){
        status = 0;
        Console.WriteLine("Đã tắt điện thoại");
    }
    public void openFile(string fileName){
        if (status == 0)
            turnOn();
        Console.WriteLine("Đã mở file " + fileName);
        isOpenAFile = 1;
    }
    public void closeFile()
    {
        if (status == 0)
            turnOn();
        Console.WriteLine("Đã đóng file đang đọc");
        isOpenAFile = 0;
    }
    public void openApplication(string applicationName)
    {
        if (status == 0)
            turnOn();
        Console.WriteLine("Đã mở ứng dụng " + applicationName);
    }
}

```

Messenger as ITarget1, MediaPlayer as ITarget2.

```

interface Messenger
{
    void sendMessageByName(string recipientName);
    void displaySentMessage();
}
interface MediaPlayer
{
    void play(string name);
    void stop();
}

```

MessengerAdapter as Adapter1, MediaPlayerAdapter as Adapter2

```

class MessengerAdapter: Smartphone, Messenger
{
    public void sendMessageByName(string recipientName){
        if (this.isOpenAFile == 1){
            this.closeFile();
        }
        this.openApplication("Nhắn tin");
        this.openFile("Danh bạ");
        Console.WriteLine("Đã gửi tin nhắn cho " + recipientName);
        this.closeFile();
    }
    public void displaySentMessage()
    {

```

```

        if (this.isOpenAFile == 1)
        {
            this.closeFile();
        }
        this.openFile("Tin nhắn lưu trữ");
        Console.WriteLine("Nội dung tin nhắn: Bạn đã trúng thưởng 100
triệu");
    }
}
class MediaPlayerAdapter: Smartphone, MediaPlayer
{
    public void play(string name)
    {
        this.openApplication("Máy nghe nhạc");
        this.openFile(name);
        Console.WriteLine("Bạn đang nghe bài hát " + name);
    }
    public void stop()
    {
        this.closeFile();
        Console.WriteLine("Đã tắt bài hát đang nghe!");
    }
}

```

Main program: Người sử dụng muốn dùng smartphone để nhắn tin và nghe nhạc.

PP1: Có sử dụng adapter

```

MediaPlayerAdapter myIphoneAsMediaPlayer = new MediaPlayerAdapter();
myIphoneAsMediaPlayer.play("Happy New Year");
myIphoneAsMediaPlayer.stop();
MessengerAdapter myIphoneAsMessenger = new MessengerAdapter();
myIphoneAsMessenger.sendMessageByName("Đặt");
Console.ReadKey();

```

PP2: Không sử dụng adapter

```

Smartphone mySmartphone = new Smartphone();
mySmartphone.turnOn();
mySmartphone.openApplication("Máy nghe nhạc");
mySmartphone.openFile("Happy New Year!");
Console.WriteLine("Bạn đang nghe bài hát Happy New Year!");
mySmartphone.closeFile();
Console.WriteLine("Đã tắt bài hát đang nghe!");
mySmartphone.openApplication("Nhắn tin");
mySmartphone.openFile("Danh bạ");
Console.WriteLine("Đã gửi tin nhắn cho Đạt!");
mySmartphone.closeFile();
mySmartphone.turnOff();
Console.ReadKey();

```

## 2. Coding convention

### 2.1. Naming convention

Cách đặt tên	Cấu trúc đặt	Ví dụ	Phạm vi sử dụng	Kiểu	Nội dung/ Ý nghĩa	Version	Pascal/ Upper case/ Camel
Class	<TênClass>	XqBoard, Game	-	-	V	-	Pascal
Interface	I<TênInterface>	IEnumerable	-	V	V	-	Pascal
Field	<KiểuBiến> <TênBiến> hoặc <KiểuBiến>_<TênBiến>	penBold, penRegular, int_IDOnCoordinate	-	V	V	-	Camel
Property	_<KiểuBiến> <TênBiến>	_hoVaTen	-	V	V	-	Camel
Variable	<TênBiến>	count	-	-	V	-	Camel
Method	(Void) <doSmth> (Bool) <isSmth> Còn lại <getSmth>	getCoordinatesByName(), drawEmptyBoard(), getPointByIntersection()	-	V	V	-	Camel
Form	<TênForm>Form	CoTuongForm, OptionsForm	-	V	V	-	Pascal
Project	<TênProject>_<Version>	MyCoTuong_102	-	-	V	V	Pascal
Solution	<TênSolution>	MyCoTuong_102	-	-	V	-	Pascal
Control	<KiểuControl_Camel> <Mô tả_Pascal>	lbNotification, tmP1Timer	-	V	V	-	Camel
File	<TênFile>.<KiểuFile>	GameConstants.cs	-	V	V	-	Camel
Constant	Trong static class <KiểuConst>_<Mô tả>	opt_gameType, res_gameType	-	V	V	-	Camel

### 2.2. Style convention

Source file	1. Không viết nhiều class trong 1 file.
-------------	---

	2. Không có nhiều namespace trong 1 file.
Ngoặc nhọn { }	3. Luôn viết trên dòng riêng
Khoảng thụt vào giữa các lớp	4. 1 Tab = 4 Spaces
Khai báo biến	5. Khai báo mỗi biến trên 1 dòng
Sử dụng kiểu dữ liệu	6. Sử dụng native data type int/string thay cho Int32/String 7. Ưu tiên sử dụng <b>int</b> khi biểu diễn số nguyên, <b>double</b> khi biểu diễn số thực. Chỉ sử dụng <b>long</b> và <b>decimal</b> trong trường hợp thực sự cần thiết. Không sử dụng các kiểu khác.
Câu lệnh điều kiện	8. Không sử dụng điều kiện phức có quá 2 lớp ngoặc
Câu lệnh phức	9. Không khai báo nhúng VD: KHÔNG dùng <code>a = 30 + (b = 15);</code>
Độ dài hàm	10. Không quá 10 dòng trừ trường hợp đặc biệt.
Đường dẫn	11. Sử dụng "@" trước mỗi địa chỉ thay vì dùng "\\".
Số lượng tham số	12. Tránh sử dụng quá nhiều tham số ( $\geq 8$ ) cho một hàm. Khi buộc phải sử dụng như vậy, định nghĩa một <b>struct</b> hoặc <b>class</b> các tham số truyền vào.
Property/Field	13. Hạn chế sử dụng field public (đặc biệt là khi biến cần truy cập nhiều). Sử dụng property thay thế.

### 2.3. Solution convention

Exception	<p>1. Không try/catch thay cho câu lệnh điều kiện. VD: <b>KHÔNG DÙNG:</b></p> <pre>try { for (int i = 0; i++)     array[i]++; } catch (ArrayIndexOutOfBoundsException e) { //do next things }</pre> <p>2. Cố gắng kiểm tra bằng câu lệnh điều kiện để tránh exception. VD <b>KHÔNG DÙNG:</b></p> <pre>try{     conn.Close(); }</pre>
-----------	--

	<pre> catch(Exception e){     //roll back } DỪNG: If (conn.State != conn.Close()){     conn.Close(); } </pre> <p>3. Chỉ sử dụng finally để giải phóng tài nguyên trong try.</p> <p>4. Khi catch exception, dẫn xuất từ lớp <b>Exception</b> thay vì <b>ApplicationException</b>.</p> <p>5. Khi không xử lý được exception thì phải throw ra ngoài.</p>
Event	6. Luôn try catch event.
Thuật toán	7. Comment đầu vào, đầu ra, mục đích sử dụng, các bước con của hàm trong trước mỗi hàm.
Sử dụng hằng số	8. Không sử dụng inline constant. Muốn sử dụng constant hãy định nghĩa nó trong một lớp riêng, ví dụ <b>GameConstants</b> .



## 3. Bài tập cá nhân

### 3.1. Cơ sở kiến thức

#### 3.1.1. Kiến thức về lập trình hướng đối tượng

- Tư duy lập trình hướng đối tượng.
- Các đặc trưng cơ bản của lập trình hướng đối tượng: tính kế thừa, tính đa hình, tính đóng gói.
- Ngôn ngữ C#, nền tảng .NET.

#### 3.1.2. Lý thuyết trò chơi: Giới thiệu về cờ tướng

##### 3.1.2.1. Mục đích của ván cờ

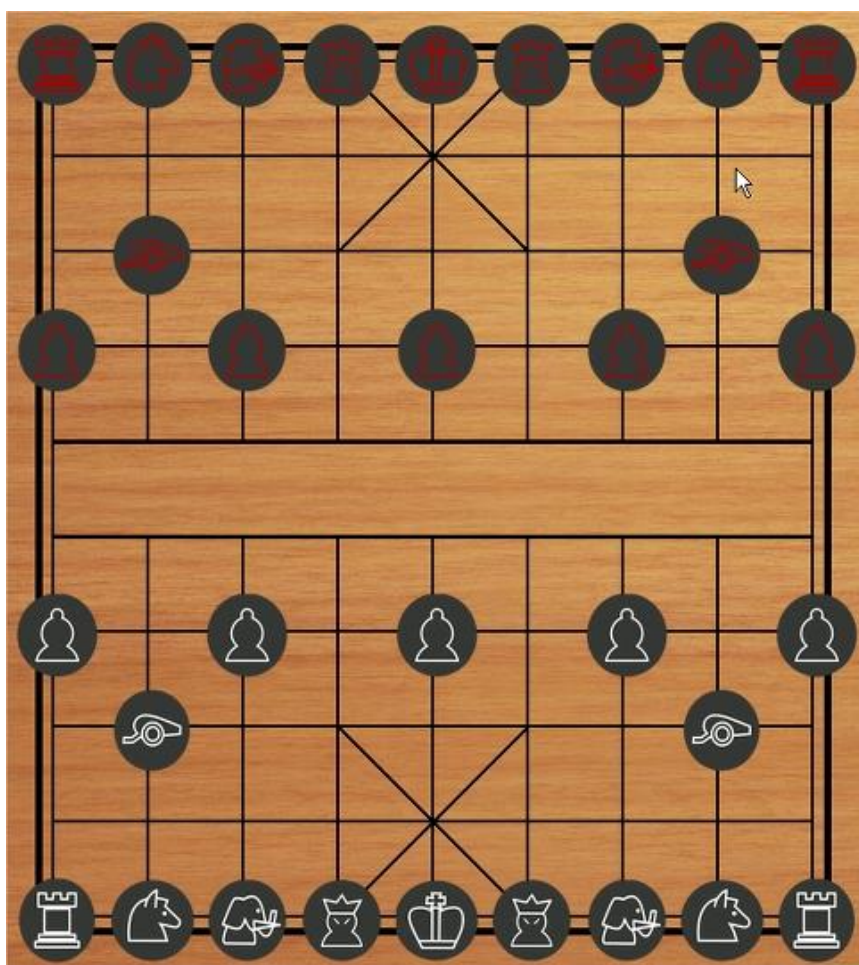
Ván cờ được tiến hành giữa hai người, một người cầm quân Đỏ, một người cầm quân Đen. Mục đích của mỗi người là tìm mọi cách di quân trên bàn cờ theo đúng luật để chiếu bí hay bắt Tướng của đối phương và giành thắng lợi.

##### 3.1.2.2. Bàn cờ và quân cờ







Bàn cờ là một hình chữ nhật do 9 đường dọc và 10 đường ngang cắt nhau vuông góc tại 90 điểm hợp thành. Một khoảng trống gọi là sông (hay hà) nằm ngang giữa bàn cờ, chia bàn cờ thành hai phần đối xứng bằng nhau. Mỗi bên có một cung Tướng hình vuông (Cửu cung) do 4 ô hợp thành tại các đường dọc 4, 5, 6 kể từ đường ngang cuối của mỗi bên, trong 4 ô này có vẽ hai đường chéo xuyên qua.




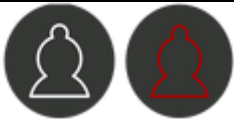
Theo quy ước, khi bàn cờ được quan sát chính diện, phía dưới sẽ là quân Đen phía trên sẽ là quân Đỏ. Các đường dọc bên Đỏ được đánh số từ 1 đến 9 từ phải qua trái. Các đường dọc bên Đen được đánh số từ 9 tới 1 từ phải qua trái.

Ranh giới giữa hai bên là "sông" (hà). Con sông này có tên là "Sở hà Hán giới" con sông định ra biên giới giữa nước Sở và nước Hán theo lịch sử Trung Quốc.



Mỗi ván cờ lúc bắt đầu phải có đủ 32 quân, chia đều cho mỗi bên gồm 16 quân Trắng (Đỏ) và 16 quân Đen. Tuy tên quân cờ của mỗi bên có thể viết khác nhau (ký hiệu theo chữ Hán) nhưng giá trị và cách đi quân của chúng lại giống nhau hoàn toàn. Bày loại quân có ký hiệu và số lượng cho mỗi bên như sau:

Quân	Tên gọi	Số lượng mỗi bên	Cách di chuyển và ăn quân
 	Tướng	1	Đi từng ô một, đi ngang hoặc dọc. Tướng luôn luôn phải ở trong phạm vi cung và không được ra ngoài.  Cung tức là hình vuông 2X2 được đánh dấu bằng đường chéo hình chữ X
 	Sĩ	2	Đi chéo 1 ô mỗi nước. Sĩ luôn luôn phải ở trong cung như Tướng.
 	Voi (Tượng)	2	Đi chéo 2 ô (ngang 2 và dọc 2) cho mỗi nước đi. Tượng chỉ được phép ở một bên của bàn cờ, không được di chuyển sang nửa bàn cờ của đối phương. Nước đi của

			tướng sẽ không hợp lệ khi có một quân cờ nằm chặn giữa đường đi.
	Mã	2	Đi ngang 2 ô và dọc 1 ô (hay dọc 2 ô và ngang 1 ô) cho mỗi nước đi. Nếu có quân nằm ngay bên cạnh mã và cản đường ngang 2 (hay đường dọc 2), mã bị cản không được đi đường đó.
	Xe	2	Đi ngang hay dọc trên bàn cờ miễn là không bị quân khác cản đường từ điểm đi đến điểm đến.
	Pháo	2	Đi ngang và dọc giống như xe. Điểm khác biệt là nếu pháo muốn ăn quân, pháo phải nhảy qua đúng 1 quân nào đó. Khi không ăn quân, tất cả những điểm từ chỗ đi đến chỗ đến phải không có quân cản.
	Tốt	5	Đi một ô mỗi nước. Nếu chốt chưa vượt qua sông, nó chỉ có thể đi thẳng tiến. Khi đã vượt sông rồi, chốt có thể đi ngang 1 nước hay đi thẳng tiến 1 bước mỗi nước.

### 3.1.2.3. Bàn cờ và quân cờ

Một bên được công nhận là thắng nếu:

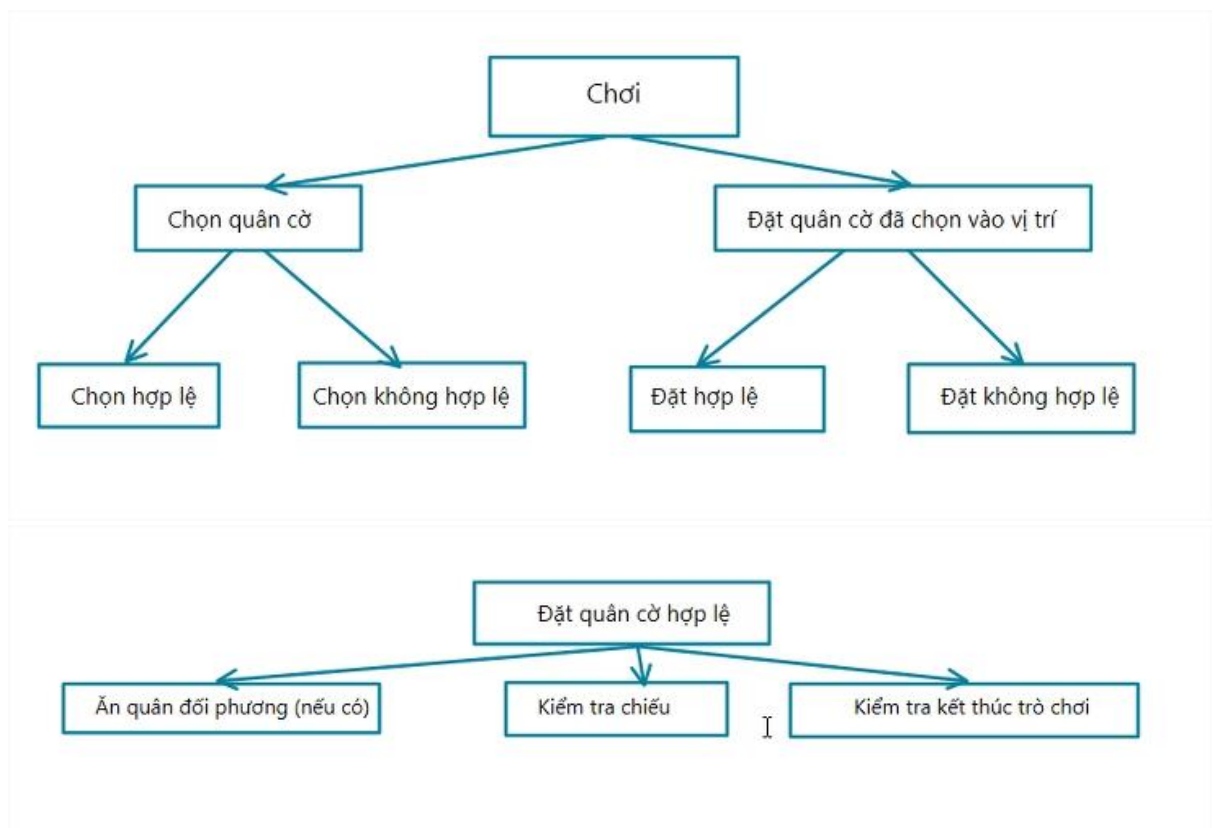
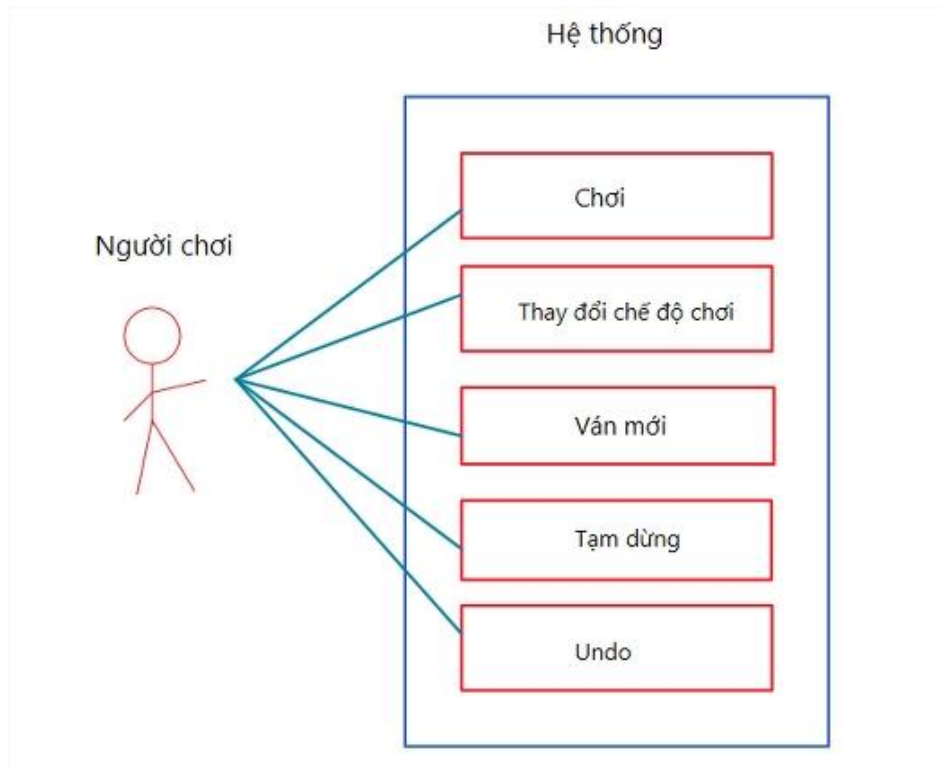
1. Chiếu bí được tướng của đối phương.
2. Đối phương hết giờ trước.
3. Đối phương chịu thua.

Một ván đấu được coi là hòa nếu:

1. Hai bên đồng ý hòa.
2. Hai bên đánh 50 nước liên tục mà không bên nào ăn được quân.

## 3.2. Tổng quan về ứng dụng

### 3.2.1. Biểu đồ use case



### 3.2.2. Đặc tả use case

Dòng điều khiển (flow control) thông thường của chương trình có dạng như sau:

[Chờ 0]

1. Người chơi ấn nút **Start**. Chương trình cho hiện bàn cờ và các quân cờ hiện lên màn hình, đồng hồ 2 bên bắt đầu chạy  
[Chờ 1]
2. Người dùng click vào một vị trí trên bàn cờ. Chương trình phân biệt xem đó là thao tác chọn quân cờ hay thao tác đi quân cờ. Nếu là chọn quân cờ đến bước 3. Nếu là đi quân cờ đến bước 4.
3. Chương trình phân tích xem người chơi có chọn đúng quân cờ không. Nếu không thì thông báo lên thanh Notification và quay lại bước [Chờ 1]. Nếu có thì thực hiện nước đi, vẽ lại trạng thái mới của bàn cờ lên màn hình.
4. Chương trình phân tích xem người chơi có đi hợp lệ hay không. Nếu không hợp lệ quay thông báo lên thanh Notification và quay lại bước [Chờ 1]. Nếu có thực hiện tiếp bước 5.
5. Thực hiện nước đi, vẽ trạng thái mới của bàn cờ lên màn hình. Kiểm tra ăn quân đối phương, kiểm tra nước chiếu, kiểm tra kết thúc trò chơi. Nếu kết thúc trò chơi xuống bước 6, còn không quay lại bước [Chờ 1].
6. Kết thúc trò chơi. Thông báo người thắng cuộc, đóng băng bàn cờ. Quay về [Chờ 0]

#### Một số ngoại lệ:

Bất cứ lúc nào người chơi cũng có thể ấn nút **Pause**, **Undo**, **New Game** trên màn hình. **Pause** sẽ đóng băng bàn cờ và thực hiện khoảng chờ ngay tại chỗ đang đứng, chờ đến khi người dùng click cho tiếp tục. **Undo** sẽ tiến hành vẽ lại trạng thái bàn cờ ở nước đi trước đó. **New Game** sẽ cho chương trình nhảy ngay lập tức đến [Chờ 0].

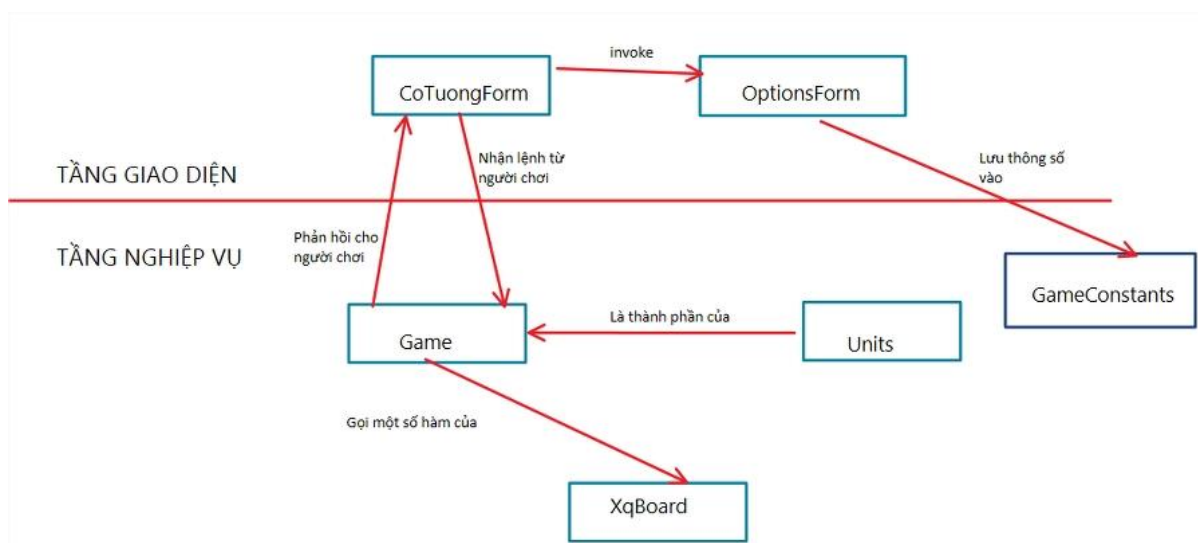
Bất cứ lúc nào người chơi cũng có thể ấn nút thoát (chữ X trên thanh cửa sổ). Chương trình giải phóng tài nguyên và thoát.

Ở đây em có 3 lớp quan trọng

- Lớp Game: Là lớp điều khiển toàn bộ tiến trình trò chơi, bao gồm xử lý các thao tác đi cờ, phản hồi tới người chơi, xử lý kiểm tra trạng thái chiếu tướng, thắng/ hòa/ thua.
- Lớp XqBoard: Là lớp bao gồm các thuộc tính và phương thức xử lý đồ họa: vẽ bàn cờ, vẽ quân cờ, highlight các nước đi.
- Unit và các lớp con của nó: lưu trữ thông tin quân cờ, thực hiện kiểm tra các nước đi về mặt logic.

### 3.3. Thiết kế chi tiết

#### 3.3.1. Sơ đồ tổng quan

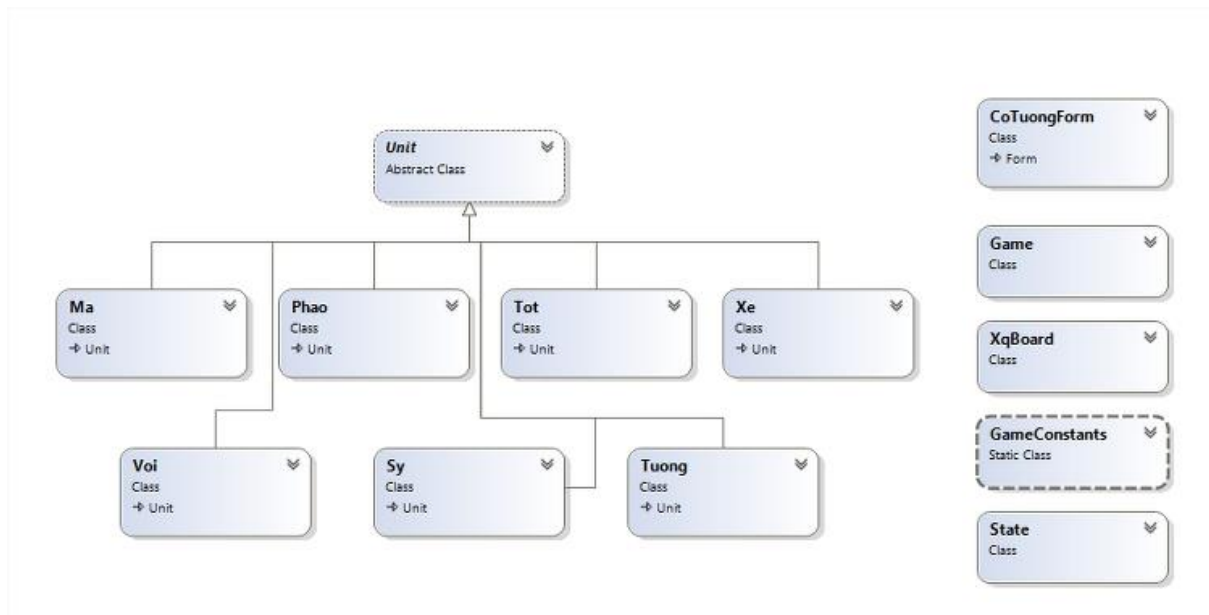


Người dùng giao tiếp với chương trình thông qua lớp **CoTuongForm**.

**CoTuongForm** nhận thao tác của người chơi rồi chuyển cho lớp **Game** xử lý. **Game** là lớp xử lý chính, bao gồm các việc: quản lý thông tin bàn cờ, vẽ bàn cờ (thông qua gọi lớp chuyên xử lý đồ họa **XqBoard**), quản lý thông tin và xử lý nước đi các quân cờ (các **Unit**). **Game** cũng phản hồi lại người chơi qua **CoTuongForm** khi cần thiết, chẳng hạn như thông báo người thắng cuộc, thông báo nước đi sai quy định,...

Khi người dùng ấn vào nút Options trên **CoTuongForm**, **OptionsForm** sẽ được mở ra để người dùng tùy chỉnh thông số trò chơi. Tất cả các thông số này được lưu vào trong **GameConstants**, một static class. Tất cả các lớp đều có quyền lấy thông số từ **GameConstants** để phục vụ xử lý của mình.

#### 3.3.2. Sơ đồ UML



*Ghi chú: Vì một số lớp có quá nhiều phương thức (10-20) nên em không cho thể hiện các phương thức của từng class trong sơ đồ này. Cụ thể có thể xem trong project.*

### 3.3.3. Lớp Game

Dưới đây em trình bày và giải thích một số xử lý quan trọng trong lớp **Game**

- Xử lý nhận thao tác đi quân cờ từ người dùng.

Hàm **handlePickOrDropEvent()** nhận vào sự kiện click chuột của người dùng, phân biệt xem đó là thao tác chọn quân cờ hay đi quân cờ rồi chuyển cho các hàm dưới xử lý.

Hàm **handlePickEvent()** thực hiện chọn quân cờ: nếu chọn hợp lệ thì lưu ID của quân cờ cần chọn. Còn nếu không thì thông báo cho người chơi.

Hàm **handleDropEventAndCheckEvent()** gọi hàm dưới để kiểm tra nước đi rồi kiểm tra xem sau nước đi đấy đã hết cờ chưa. Nếu hết cờ rồi thì kết thúc ván đấu.

```

public void handlePickOrDropEvent(MouseEventArgs e)
{
    /*Gồm các bước sau:
    * 1. Lấy tọa độ (in pixel) mà user vừa click.
    * 2. Chuyển tọa độ in pixel thành tọa độ bàn cờ.
    * 3. Thực hiện chọn quân cờ (hoặc đặt quân cờ).
    */
    Point pInPixel = e.Location;
    Point pInCoordinates = xqBoard.getCoordinatesByLocation(pInPixel);
}
  
```

```

        if (int_currentlySelected == GameConstants.unselected)
        {
            handlePickEvent(pInCoordinates);
        }
        else
        {
            handleDropEventAndCheckGameEnd(pInCoordinates);
        }
    }
    private void handlePickEvent(Point prm_pInCoordinates)
    {
        try
        {
            Unit u = getPickedUnitByCoordinates(prm_pInCoordinates);
            int_currentlySelected = u.int_ID;
            pbBoard.Invalidate();
        }
        catch (NullReferenceException expNullRef)
        {
            MyCoTuongForm.lbNotificationSetText("Bạn chưa chọn quân cờ");
        }
        catch (Exception e) {
            MyCoTuongForm.lbNotificationSetText("Đã có lỗi xảy ra khi chọn quân cờ");
            throw e;
        }
    }
    private void handleDropEventAndCheckGameEnd(Point prm_pInCoordinates)
    {
        processMove(prm_pInCoordinates);
        if (isGameEndByOneSideHasNoMove())
        {
            endGame();
        }
    }
}

```

- Thực hiện nước đi và xử lý các vấn đề liên quan

```

private void processMove(Point pInCoordinates)
{
    /*Gồm các bước sau:
    * 1. Kiểm tra xem nước đi có phù hợp không.
    * 2. Nếu nước đi phù hợp, thực hiện:
    *     2.1. Ăn quân (nếu có)
    *     2.2. Update vị trí mới cho quân cờ vừa đi
    *     2.3. Kết thúc lượt, update lượt và đổi đồng hồ
    *     2.4. Kiểm tra xem có bên nào bị chiếu không?
    *     2.5. Kiểm tra xem có hòa theo luật 50 nước không ăn quân
    không?
    *     2.6. Lưu trạng thái bàn cờ
    * 3. Nếu nước đi không hợp lệ, thông báo tại sao không phù hợp*/
    Unit u = unitOnBoard[int_currentlySelected];
    if (u.int_side == int_turn)
    {
        if (u.checkNextMove(u.pCurrentLocation, pInCoordinates))
        {
            removeOldUnitIfExists(pInCoordinates);
        }
    }
}

```



```

        updateNewPosition(u, pInCoordinates);
        updateTurnAndTimer();
        pbBoard.Invalidate();
        int_currentlySelected = GameConstants.unselected;
        processChecked();
        tieCheck();
        int_NoOfTurnWithoutUnitLoss++;
        saveState();
    }
    else notifyInvalidMove();
}
else notifyNotYourTurn();
}

```

Hàm **processMove()** nhận vào 1 tọa độ trên bàn cờ, đầu tiên kiểm tra quân cờ đang được chọn có phải quân cờ của bên đang lượt đi hay không, nếu không thì phản hồi lại chưa đến lượt. Nếu có thì tiếp tục kiểm tra xem quân cờ đang được chọn đi đến điểm đó có hợp lệ không. Nếu có thì tiến hành: ăn quân (nếu có), cập nhật lại vị trí quân cờ, đổi lượt, đổi đồng hồ, bỏ chọn quân cờ, kiểm tra chiếu tướng, kiểm tra điều kiện hòa, lưu trạng thái bàn cờ.

- Kiểm tra xem đã hết cờ chưa và kết thúc game.

```

public bool isGameEndByOneSideHasNoMove()
{
    bool ended = true;
    int startIndex = GameConstants.blackMinimumID;
    if (int_turn == GameConstants.redSide)
    {
        startIndex = GameConstants.redMinimumID;
    }
    for (int i = startIndex; i < startIndex +
GameConstants.blackMinimumID; i++){
        if (!unitOnBoard[i].bo_isAlive) continue;
        if (isAbleToMove(unitOnBoard[i])){
            ended = false;
        }
    }
    return ended;
}

public void endGame()
{
    /* 1. Dừng đồng hồ
    * 2. Tính xem người chơi nào thắng (hoặc hòa). Người thắng là
    người đang không trong lượt.
    * 3. Thông báo người chiến thắng.
    */
    string str_winner;
    //1
    MyCoTuongForm.redTimerStop();
    MyCoTuongForm.blackTimerStop();
    //2
    if (intWinner == GameConstants.neitherSide)
    {

```

```

        MyCoTuongForm.lbNotificationSetText("Hòa.");
        return;
    }
    intWinner = (int_turn + 1) % 2;
    if (intWinner == GameConstants.redSide) str_winner = "Đỏ";
    else str_winner = "Đen";
    //3
    MyCoTuongForm.lbNotificationSetText(str_winner + " thắng.");
    isEnded = GameConstants.EndState;
}

```

Hàm **isGameEndByOneSideHasNoMove()** kiểm tra xem bên có lượt tiếp theo còn nước có thể đi nữa không. Nếu không thể đi quân nào tức là đã phân định thắng thua, hàm trả về true.

Hàm **endGame()** thực hiện các thao tác kết thúc trò chơi: dừng đồng hồ, thông báo người chiến thắng, chỉnh trạng thái chơi thành kết thúc.

### 3.3.4. Lớp XqBoard

Dưới đây em xin trình bày và giải thích một số xử lý trong lớp **XqBoard**

- Vẽ bàn cờ

```

public void drawXqBoardAndSetupIntersection(PaintEventArgs e)
{
    /*Vẽ bàn cờ. Gồm các công việc:
    * 1. Thiết lập tọa độ cho các giao điểm
    * 2. Vẽ các đường
    */
    setCoordinates();
    drawXqBoardLines(e);
    drawLLine(e);
    //drawChineseBoundary(e);
}

private void setCoordinates()
{
    //Đặt tọa độ các giao điểm trên bàn cờ dựa trên tọa độ thực tế.
    for (int i = 0; i < GameConstants.boardWidth; i++)
    {
        for (int j = 0; j < GameConstants.boardHeight; j++)
        {
            pCoordinates[i, j] = new Point(intOriginalX + int_sLength *
i, intOriginalY + int_sLength * j);
        }
    }
}

```

Hàm **drawXqBoardAndSetupIntersection** thực hiện việc vẽ bàn cờ (trống không) và thiết lập hàm đổi tọa độ: từ tọa độ bàn cờ ra tọa độ thực tế, thông qua hàm **setCoordinates()**.

### - Highlight các giao điểm

```
public void highlightSelectedUnit(PaintEventArgs e, int selectedID, Color
color)
{
    //Tìm tọa độ quân cờ cần highlight rồi highlight
    Unit u = Game.unitOnBoard[selectedID];
    Point p = u.pCurrentLocation;
    int x = p.Coordinates[p.X, p.Y].X;
    int y = p.Coordinates[p.X, p.Y].Y;
    Rectangle rec = new Rectangle(x - int_sLength * 5 / 9, y -
int_sLength * 5 / 9, int_sLength * 9 / 8, int_sLength * 9 / 8);
    e.Graphics.FillEllipse(new SolidBrush(color), rec);
}

public void highlightAvailableMove(PaintEventArgs e, int selectedID,
Color color, int alpha)
{
    //Highlight các vị trí có thể đi được
    Unit u = Game.unitOnBoard[selectedID];
    for (int i = 0; i < GameConstants.boardWidth; i++)
    {
        for (int j = 0; j < GameConstants.boardHeight; j++)
        {
            if (u.checkNextMove(u.pCurrentLocation, new Point(i, j)))
            {
                highlightOnPosition(e, p.Coordinates[i, j].X,
p.Coordinates[i, j].Y, Color.FromArgb(alpha, 152, 251, 152));
            }
        }
    }
}

private void highlightOnPosition(PaintEventArgs e, int x, int y, Color
color)
{
    //Highlight 1 vị trí nào đó bởi 1 màu nào đó
    Rectangle rec = new Rectangle(x - int_sLength * 3 / 9, y -
int_sLength * 3 / 9, int_sLength * 5 / 8, int_sLength * 5 / 8);
    e.Graphics.FillEllipse(new SolidBrush(color), rec);
}
```

Hàm **highlightSelectedUnit()** highlight quân cờ đang được chọn bằng một màu nào đó. Hàm **highlightAvailableMove()** highlight các vị trí có thể đi được của quân cờ đang được chọn. Hàm **highlightOnPosition()** sẽ highlight một vị trí bất kì bằng một màu được chỉ định.

Ở đây em không đưa các giá trị 3/9, ... vào **GameConstants** vì có quá nhiều giá trị, mà trong tương lai không cần sửa đến nó một lần nào nữa.

### 3.3.5. Lớp trừu tượng Unit và các lớp con

Unit là lớp trừu tượng chứa thông tin của các quân cờ và các phương thức kiểm tra nước đi của quân cờ đó

```

abstract class Unit
{
    public int int_ID;
    //Each unit on boards have a unique ID (0-31)

    public int int_side;
    //0: red, 1: black

    public string str_name;

    public Point pCurrentLocation;
    //represent the real location (by pixels) on board

    public bool bo_isAlive = true;
    //TRUE: alive, FALSE: dead

    public Image img_unitImg;
    //Image displayed on board

    private int int_oldID = GameConstants.noID;

    //Kiểm tra xem nước đi tiếp theo có hợp lệ?
    public virtual bool checkNextMove(Point currentPos, Point nextPos)
    {
        int x1 = currentPos.X;
        int y1 = currentPos.Y;
        int x2 = nextPos.X;
        int y2 = nextPos.Y;
        return true;
    }
    protected bool isGameLostAfterThisMove(int x1, int y1, int x2, int y2)
    {
        temporarilyPut(x1, y1, x2, y2);
        if (Game.isGameChecked(int_side))
        {
            rollBack(x1, y1, x2, y2);
            return true;
        }
        else rollBack(x1, y1, x2, y2);
        return false;
    }
    protected void temporarilyPut(int x1, int y1, int x2, int y2)
    {
        //Thử đặt quân cờ ở vị trí (x2, y2) xem có khiến mình bị thua hay
không?

        //Bỏ quân cờ hiện đang ở (x2, y2) (nếu có) ra ngoài.
        int_oldID = Game.int_IDOnCoordinates[x2, y2];
        if (int_oldID != -1)
        {
            Game.unitOnBoard[int_oldID].bo_isAlive = false;
        }

        //Bỏ quân cờ này khỏi (x1, y1)
        Game.int_IDOnCoordinates[x1, y1] = GameConstants.noID;
        this.pCurrentLocation = new Point(x2, y2);

        //Ghi nhận quân cờ này ở (x2, y2)
        Game.int_IDOnCoordinates[x2, y2] = this.int_ID;
    }
    protected void rollBack(int x1, int y1, int x2, int y2)
    {
        //Đặt quân cờ trở lại điểm (x1, y1). Đặt trả lại trạng thái quân cờ
ở điểm (x2, y2).

```

```

        this.pCurrentLocation = new Point(x1, y1);
        Game.int_IDOnCoordinates[x1, y1] = this.int_ID;
        Game.int_IDOnCoordinates[x2, y2] = int_oldID;
        if (int_oldID != GameConstants.noID)
            Game.unitOnBoard[int_oldID].bo_isAlive = true;
        int_oldID = GameConstants.noID;
    }
    public bool isAbleToWinFirst(int x2, int y2)
    {
        Unit u;
        if (this.int_side == 0) u =
Game.unitOnBoard[GameConstants.blackMinimumID];
        else u = Game.unitOnBoard[GameConstants.redMinimumID];
        if (u.pCurrentLocation.X == x2 && u.pCurrentLocation.Y == y2)
return true;
        return false;
    }
}

```

Các trường **ID**, **int\_side**, **str\_name**, **pCurrentLocation**, **bo\_isAlive**, **img\_unitImg** lần lượt chứa thông tin ID, màu, tên, vị trí hiện tại, còn sống/đã bị ăn, hình ảnh của quân cờ. Hàm **checkNextMove()** là virtual chờ được override bởi từng lớp con tương ứng. Hàm **isGameLostAfterThisMove()** thử đi quân cờ từ vị trí (x1, y1) đến (x2, y2) để xem nước đi đó có làm bên ta thua được không, thực thi qua hai hàm dưới **temporarilyPut()** và **rollBack()**.

#### - Cách thức đi của con Mã

```

public Ma(int ID)
{
    //this.Game = g;
    this.ID = ID;
    this.int_side = (ID) / 16;
    this.str_name = "Mã";
    int x = (ID % 2 == 1) ? 1 : 7;
    int y = (int_side == 0) ? 0 : 9;
    this.pCurrentLocation = new Point(x, y);
    string s = (int_side == 0) ? "do":"den";
    this.img_unitImg = Image.FromFile(GameConstants.res_UnitImg + "ma"
+ s + "3.png");
}
    public override bool checkNextMove(Point cor_currentPos, Point
cor_nextPos)
    {
        /*Gồm các bước:
        * 1. Kiểm tra xem mã đi có đúng đường không?
        * 2. Kiểm tra xem mã có bị chặn không?
        * 3. Kiểm tra xem mã có ăn quân bên mình không?
        * 4. Kiểm tra xem tướng có nguy hiểm hay không?
        */
    }
}

```

```

int x1 = cor_currentPos.X;
int y1 = cor_currentPos.Y;
int x2 = cor_nextPos.X;
int y2 = cor_nextPos.Y;
//1
//
int s = Math.Abs(x1 - x2) + Math.Abs(y1 - y2);
int d = Math.Abs(Math.Abs(x1 - x2) - Math.Abs(y1 - y2));
if (s != 3 || d != 1)
    return false;
//2
if (isBlockedHorse(x1, y1, x2, y2)) return false;
//3
if (Game.isPossed(x2, y2) == int_side) return false;
//4
if (isAbleToWinFirst(x2, y2)) return true;
if (isGameLostAfterThisMove(x1, y1, x2, y2)) return false;
return true;
}
public bool isBlockedHorse(int x1, int y1, int x2, int y2)
{
    if (Math.Abs(x1 - x2) == 1)
    {
        if (y2 > y1)
        {
            if (Game.isPossed(x1, y1 + 1) == int_side) return true;
        }
        else if (Game.isPossed(x1, y1 - 1) >= 0) return true;
    }
    else if (Math.Abs(y1 - y2) == 1)
    {
        if (x2 > x1)
        {
            if (Game.isPossed(x1 + 1, y1) == int_side) return true;
        }
        else if (Game.isPossed(x1 - 1, y1) >= 0) return true;
    }
    return false;
}
}

```

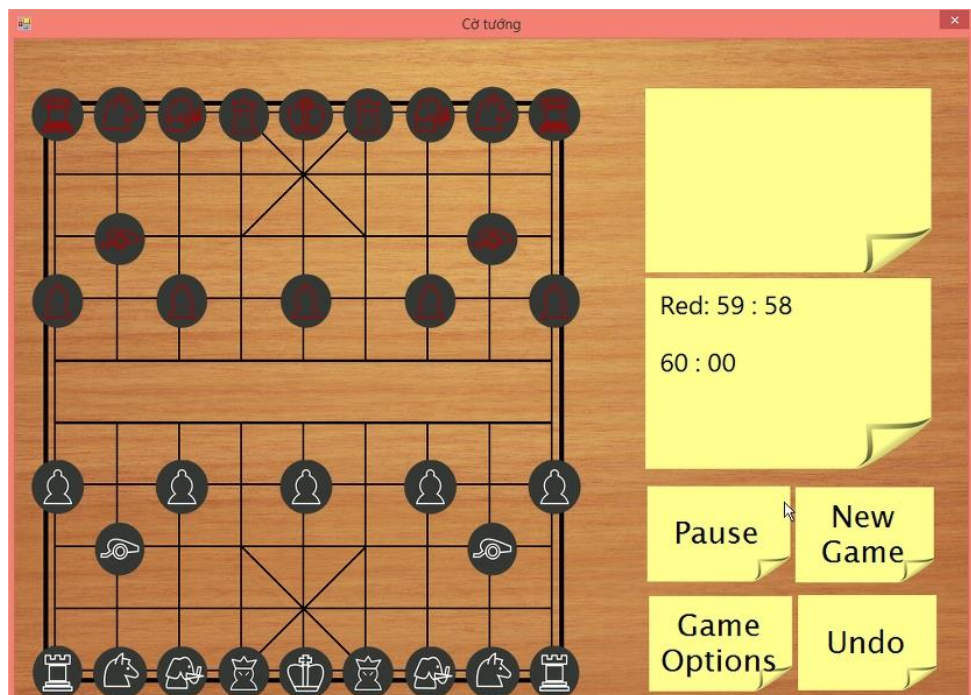
Hàm **checkNextMove()** được override, nó kiểm tra tính hợp lệ của nước đi thông qua các tiêu chí: mã có đi đúng đường không, mã có bị chặn không, mã có “ăn” quân bên mình không, mã đi như vậy có khiến bên mình bị thua luôn hay không.

### 3.4. Kết quả chương trình

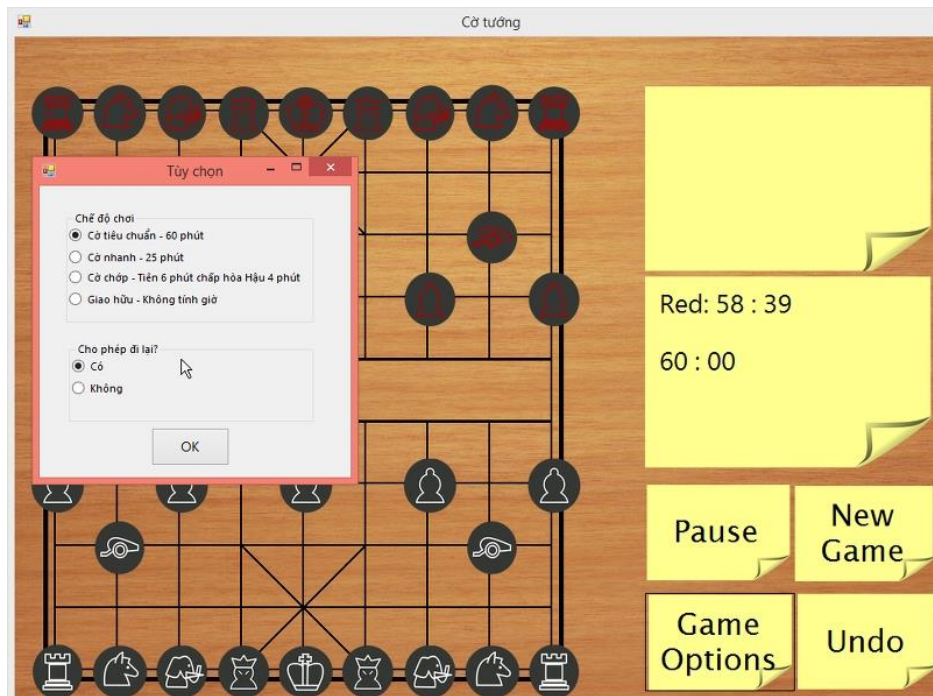
Quá trình lập trình bắt đầu từ khi em viết dòng code đầu tiên cho đến nay đã được gần 2 tháng. Quãng thời gian này tuy không dài nhưng cũng đủ để em viết được chương trình trò chơi cờ tướng hoàn chỉnh và thì giờ để test thử tương đối kĩ. Chương trình mới là 2 người chơi với nhau còn khá đơn giản, chưa có nhiều tính năng độc đáo. Nếu có điều kiện phát triển thêm, em sẽ nghiên cứu trí tuệ nhân tạo để thêm vào tính năng đánh người – máy, tính năng tạo giải đấu, tạo trận đấu giống như ngoài đời (1 ván cờ tiêu chuẩn hòa → 1 ván cờ nhanh hòa → 1 ván cờ chớp để phân định thắng thua). Đồng thời cũng nâng cấp giao diện cho đẹp hơn.

Dưới đây là minh họa một số màn hình chức năng chính của chương trình

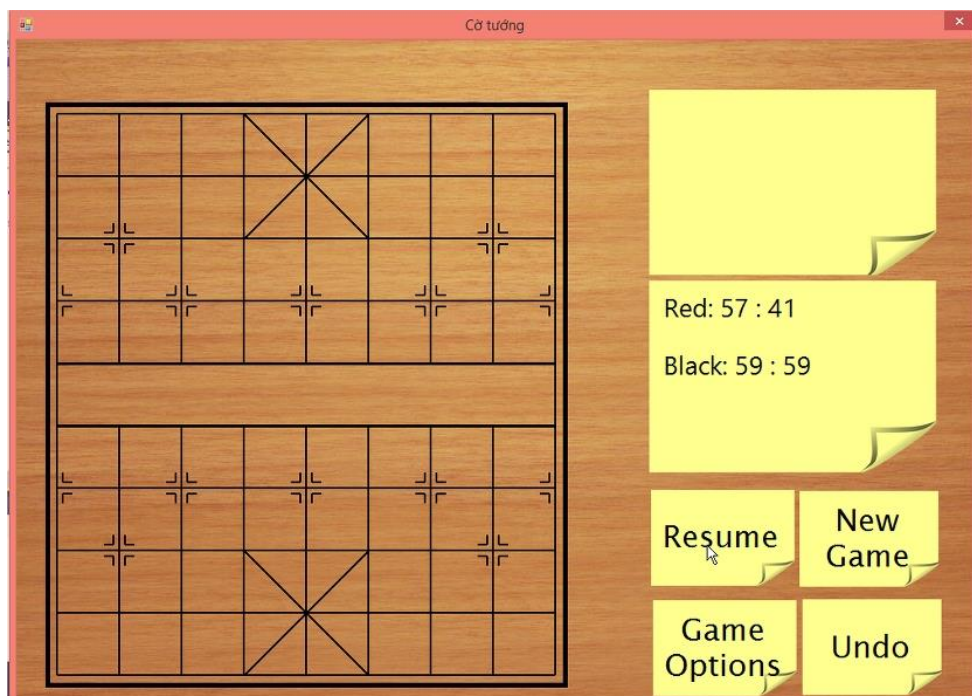
Khi đang chơi



GameOptions – chỉnh sửa thông số

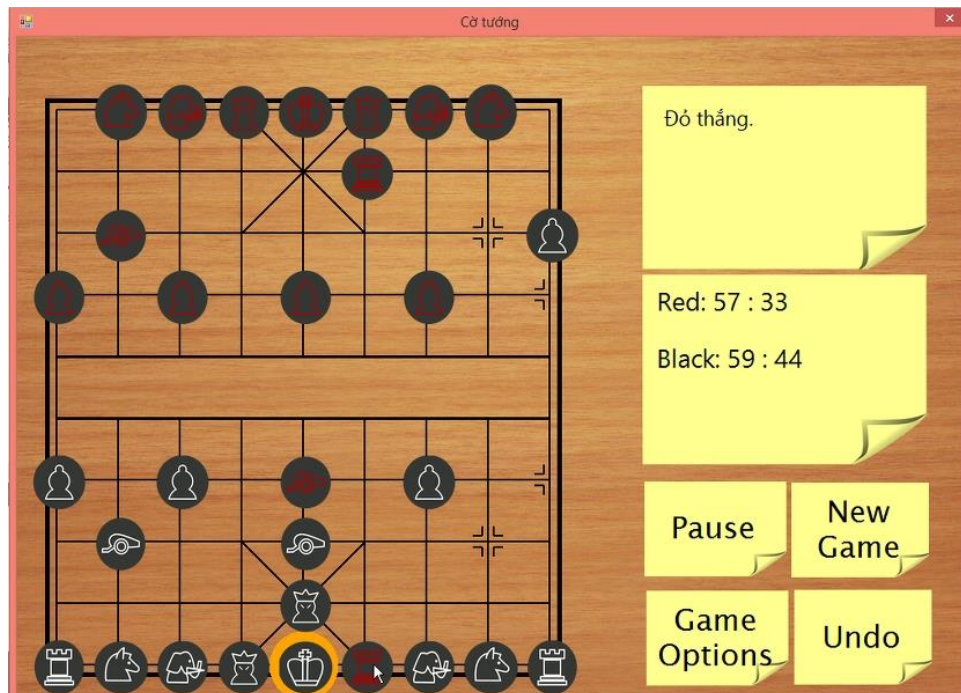


Khi tạm dừng game





Và Đỏ đã thắng!



## Phụ lục: Tài liệu tham khảo

1. [Tập hợp các bài giảng .NET của thầy Tú ở BKIndex](#)
2. C# 3.0 Design Patterns, Judith Bishop
3. [Microsoft Developer Network](#)
4. [StackOverFlow](#)