LỜI NÓI ĐẦU

Ước mơ viết 1 game cho riêng mình , rộng ra thì viết 1 game cho VN mở mặt với thế giới. Ước mơ của mình là muốn VN chúng ta có một ngành CN giải trí điện tử lớn mạnh ngang tầm tụi Hàn chẳng hạn và rồi sẽ có ngày bọn tây, tàu đến VN xin mua bản quyền game đem về nc nó phát hành lại ɹ .

Hiện nay có vô số công cụ, Ngôn ngữ, môi trường phát triển game, mình chọn XNA bởi các lý do sau:

-Các ưu điểm đã nêu trong forum, trên internet...

-Của Microsoft mà mình tín nhiệm thằng này.

-Môi trường làm việc chuyên nghiệp, quản lý tốt. free khi làm game trên windows. Khỏi lo sau này lôi kéo nhau ra toà.

-NNLT C# đơn giản, tư nhiên, dễ học và hiểu nhanh hơn so với các NNLT khác.

Bài tut này nhằm mục đích nâng cao tay nghề cho bản thân và mình muốn phổ biến những kiến thức cơ bản nhất về XNA cho mọi người. Sau này nếu có các dự án lớn trên forum thì chắc mọi người đều là các programmer online cả rồi. Tương lai là điều rất khó đoán, thế nên ta nên học từ ngay bây giờ đi, sau này nhất định sẽ có lúc cần, nhất là các bạn theo IT và tính cả các bạn đam mê IT nữa. Một game Nhập vai trực tuyến nhiều người chơi nếu chỉ dựa vào tut này thì ko thể nhưng viết những game nhỏ tặng bạn bè, bồ bịch hay chuẩn bị cho những dự án lớn hơn (có thể trong tương lai các bạn là người mình muốn cộng tác đấy) thì mình tin tài liệu này sẽ có ích.

Đối với nhiều người, lập trình và công việc nhàm chán, tắm mình trong 1 đống code, thế nhưng muốn viết game phải biết lập trình, bạn hãy nhớ công đoạn thú vị nhất khi lập trình game là lập trình chúng (ngoài ra LT ra còn có viết ý tưởng, thuật toán, debug...) Lập trình không chỉ là công việc đó là nghệ thuật còn Lập trình viên là nghệ sĩ, tất nhiên rồi :D

Bài viết dựa trên tài liệu: *Beginning XNA 3.0 Game Programming From Novice to Professional.* ***nhà xuất bản APRESS.*** Bài viết không thể tránh khỏi sai sót, mọi gọi ý thắc mắc kiện cáo j, các bạn bỏ hết vào đây thanh_vinh648@yahoo.com (nếu bận mình ko online thường xuyên, các bạn có thể dùng số DĐ sau: 01649120185-Nghiêm cấm nhá máy ɹ)

Tác Giả

Vũ Thành Vinh (Huyết sát)

Giới thiệu sơ lược về XNA và C#:

XNA

Không cần giới thiệu chi cho rườm rà, bạn chỉ cần có Visual Studio và bộ XNA GSE 3.0 là ok.

Chuẩn bị môi trường phát triển :

- Nâng cấp Visual Studio :

Các phiên bản VS được XNA hỗ trợ :

Trích dẫn:

> * Visual C# 2005 Express Edition
> * Visual Studio 2005 Standard Edition
> * Visual Studio 2005 Professional Edition
> * Visual Studio 2005 Tools for the Microsoft Office System
> * Visual Studio 2005 Team Edition for Software Architects
> * Visual Studio 2005 Team Edition for Software Developers
> * Visual Studio 2005 Team Edition for Software Testers
> * Visual Studio 2005 Team Edition for Database Professionals
> * Visual Studio 2005 Team Suite

+ Nếu bạn đang xài VS C# Express thì down cái này (24.3MB) :

Trích dẫn:

> http://download.microsoft.com/downlo...9-X86-INTL.exe

Nếu ko xài C# mà xài các ngôn ngữ khác thì xem thêm tại

Trích dẫn:

> http://www.microsoft.com/downloads/d...ng=en#filelist

)

+ Nếu bạn đang xài các phiên bản VS khác (ko phải Express) thì down cái này (431.7MB - khủng khiếp T_T):

Trích dẫn:

> http://download.microsoft.com/downlo...01-X86-ENU.exe

- Tiếp đó là download XNA (98.6MB):

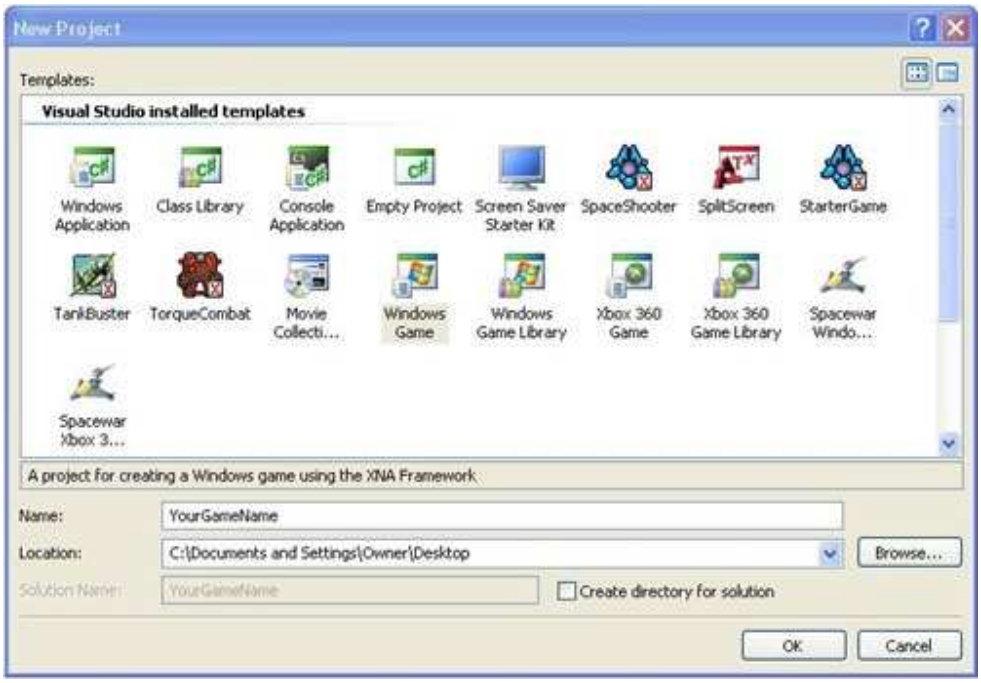
Trích dẫn:

> http://download.microsoft.com/downlo...GS20_setup.exe

Nếu đọc được tiếng Anh, bạn có thể tham khảo phần hướng dẫn của MSDN

Làm quen với XNA

Bây giờ ta sẽ tiến hành tạo một XNA Project mới
Khởi động C#, bạn chọn New Project -> Windows Game , rồi gõ tên Project vào, nhấn OK là tạo xong :-D



Bạn build ra rồi chạy thử sẽ được như hình :

=> Ra cái cửa sổ xanh lè và không có con chuột là OK!

Nguồn: ko biết nữa nhưng thấy đầy rẫy trên mạng,

Thanks tác giả.

Để hiểu mã lệnh bắt buộc các bạn phải học qua C#, tự học, online j cũng đc, mình không giải thích nhiều về C#,mà chủ yếu là các thành phần của XNA. Thường những cái đã giải thích , lession sau mình bỏ qua hoặc nói sơ sơ...

Nội dung bài viết chủ yếu về thuật toán và mã lệnh

File mã lệnh gồm có Game1.cs và prógram.cs và những thứ tự viết...

Nội dung game1.cs:

Đây là file chứa những mã lệnh giúp 1 game có thể chạy đc bình thường, tất nhiên cấu thành nó gồm nhiều Component và DrawableGameComponent (class sẵn có trong XNA) ...

Đây là giới thiệu cơ bản, tuy đơn giản nhưng các bạn nên đọc qua >= 1 lần

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
//Những thứ XNA sử dụng bạn ko cần using hết mà tuỳ từng trường hợp mà sử dụng

namespace WindowsGame1
{
    /// <summary>
    /// Đây là những thiết lập ban đầu cho 1 SpriteBatch đặc biệt gọi là SpriteBatch và đồ họa, tất cả đều dựa trên mã nguồn của XNA
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }

        /// <summary>
        /// Cho phép game có thể bắt đầu thiết lập truớc khi bắt đầu chạy
        /// Đây là nơi bạn khai báo biến, load Tài nguyên, hàm không fải đồ họa (đồ họa sẽ có hàm khác load) ...
        /// Thiết lập cho các thành phần của game
        /// </summary>
```

```
        protected override void Initialize()
        {
            // Thêm vào thiết lập logic ở đây, thường cũng k cần lắm

            base.Initialize();
        }

        /// <summary>
        /// hàm LoadContent sẽ được gọi 1 làn khi chạy game và nó sẽ load
toàn bộ những content trong game của bạn
        /// </summary>
        protected override void LoadContent()
        {
            // tạo một SpriteBatch mới, nó được dùng để vẽ texture.
            spriteBatch = new SpriteBatch(GraphicsDevice);

                    }

        /// <summary>
        /// hàm UnloadContent đc gọi 1 làn trong game khi bạn out
        /// nó xóa sạch nội dung chứa trong game
        /// </summary>
        protected override void UnloadContent()
        {
            // không bắt buộc fải có, trừ dự án lớn hoặc khi bạn siêng năng.
        }

        /// <summary>
        /// hàm Update() hàm kt ĐK của game khi chạy
        /// kiểm tra  kiểm tra  va chạm, input (mouse, keyboard), và chơi
nhạc
             protected override void Update(GameTime gameTime)
        {
            // cho phep game out (trên Xbox)
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)
                this.Exit();

            // thêm vào logic của game ở đây

            base.Update(gameTime);
//GameTime là 1 biến đặc biệt chỉ thời gian chạy game
        }

        /// <summary>
        /// Hàm gọi khi cần vẽ thì hàm sẽ hiện lên
        /// </summary>
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
//xóa màn hình và thêm màn nền (Color.CornflowerBlue)
            // thêm vào mã lệnh đồ họa.

            base.Draw(gameTime);
        }
    }
}
```

Đây là nội dung cơ bản của XNA, Nhiệm vụ của nó là hiện lên cái màn hình xanh lè:

Trong đây chỉ có hàm `Draw() là thể hiện`

File mã lệnh thứ 2 là program.cs:

```
using System;

namespace XNADemo
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
}
```

//cái nài thì cứ để yên thế, không nên đụng tay vô J

Các file mã lệnh: class do user tự Đ/N, cái này do mình viết và nó ko hề ít đâu L

Bài 1 : Đưa Đối tượng lên Màn hình

Tài nguyên: một texture (image) 64x64 có tên ball

Đầu tiên khi viết game là Object phải được đưa lên screen, nghe thì đơn giản nhưng trong XNA đây là việc kha khá đâu đầu, đòi hỏi sự phối hợp giữa các hàm trong Game1.cs

Đầu tiên bạn khởi taọ project mới.cho nó 1 tài nguyên (image chẳng hạn)

Tại cửa sổ project, bạn click phải chuột vào mục **Content**, chọn **Add -> Existing Item**, hộp thoại Add Existing Item mở ra và bạn tìm đến file cần đưa vào project, add nó vào

Phần mã lệnh:

I Một đối tượng đưa lên monitor thì cần các yếu tố sau:

SpriteBatch : một Sprite đặc biệt , nó đại diện cho một hoặc các sprite khác trong XNA.

Nó có sẵn hay tự tạo đều được.

Texture: Đại loại là một bức ảnh 2D (cũng có texture3D) bao phủ lên Sprite trong game (thường là bao theo hình chữ nhật).

Class: lớp là thành phần không thể thiếu của C# vì đây là ngôn ngữ hướng đối tượng hoàn toàn, bất kẻ sprite nào muốn thể hiện điều phải có class rieng của nó.

Phần I: class cho clsSprite

Tính hợp thành của OOP (**Composition)**: Tất cả object chạy trên chương trình đều có mã lệnh hoạt động riêng của nó đc viết lại thành các hàm, method, thuộc tính trong class của nó, tất cả chúng hợp thành lên class cho object. Nó có thể thừa kế từ nhiều nguồn khác nhau, sau này sẽ nói thêm.

Tạm thế đã, đầu tiên chúng ta đi từ cái đơn giản nhất, mã lệnh cho một sprite đơn giản thể hiện trên màn hình:

Tạo một file *.cs đặt trong thư mục chứa project, hay add => class cung đc

Câu lệnh using sử dụng code sẵn có của XNA:

```
using Microsoft.Xna.Framework.Graphics;   //   for Texture2D
using Microsoft.Xna.Framework;  //  for Vector2
```

Cấu trúc:

Namespace => class =>function...

Sau class là phần tên lớp, đặt jì là tùy bạn, mình lấy clsSprite

```
class clsSprite
    {

    }
```

1.Khai báo

Sau đó là phần khai báo biến, cũng là các thành phần của class:

Vector2 là cái j?, nó là 1 cặp (2 cái) biến có liên quan đến nhau trong game, bạn nhận thấy vị trí của vật gồm có thuộc tính X và Y, tốc độ của sprite cũng thế có thể theo trong Ox hoac Oy, (như SpeedX, SpeedY)... tùm lum thứ có thể VD.

Do đó, chúng ta sử dụng vector2 cho các thuộc tính size (height, width)

Tương tự cho tọa độ position (X,Y)

Texture thì bắt buộc rồi:

```
        public Texture2D texture { get; set; }
//  sprite texture, read-only property
        public Vector2 position { get; set; }
//  sprite position on screen
        public Vector2 size { get; set; }

//  sprite size in pixel
```

2.Nhập dữ liệu

Một hàm trùng tên với class sẽ đảm nhận vai trò này: rất đơn giản thôi, nó nhận texture, position và size của từ bên ngoài làm của nó

```
public clsSprite (Texture2D newTexture, Vector2 newPosition, Vector2
newSize){
            texture = newTexture;
            position = newPosition;
            size = newSize;
        }
```

3.Hàm đồ họa

Nhiệm vụ của sprite là thể hiện trên screen, thế nên hàm chức năng của nó chỉ cần Draw là đủ:

```
public void Draw(SpriteBatch spriteBatch)
```

```
        {
            spriteBatch.Draw(texture, position, Color.White);

        }
```

Các yếu tố cơ bản như texture, vị trí, màu khi vẽ nhưng texture sẽ đè lên cái White đó, bạn khỏi lo, thêm mấy cái { nữa là xong class...

Phần II: Hoàn thiện cho Game1.cs:

Game1 sử dụng:

```
#region Using Statements
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using System.Linq;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Media;
using Microsoft.Xna.Framework.Net;
#endregion
```

Cứ bỏ hết vào cũng đc, thừa > thiếu

Game1 kế thừ từ `Microsoft.Xna.Framework.Game`

Class Game1 cần đc bổ sung, bắt đầu là khai báo các biến sẽ dùng:

```
        GraphicsDeviceManager graphics;
        clsSprite mySprite1;
        SpriteBatch spriteBatch;
```

Một đồ họa, sử dụng cái sẵn có của XNA

Mổt Spite, cái chúng ta đã viết class.

SpriteBatch, cái để đại diện sprite.

Game1 không cần nhập dữ liệu từ bên ngoài như clsSprite nhưng chúng ta vẫn viết hàm Game1() để thiết lập cho nó:

```
public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            //  Sử dụng đồ họa cho chính nó (game1)
            graphics.PreferredBackBufferWidth = 500;
            graphics.PreferredBackBufferHeight = 300;
            //Chiều rộng cửa sổ game
            Content.RootDirectory = "Content";
```

```
            //Thư mục gốc chứa tài nguyên là “content”
        }
```

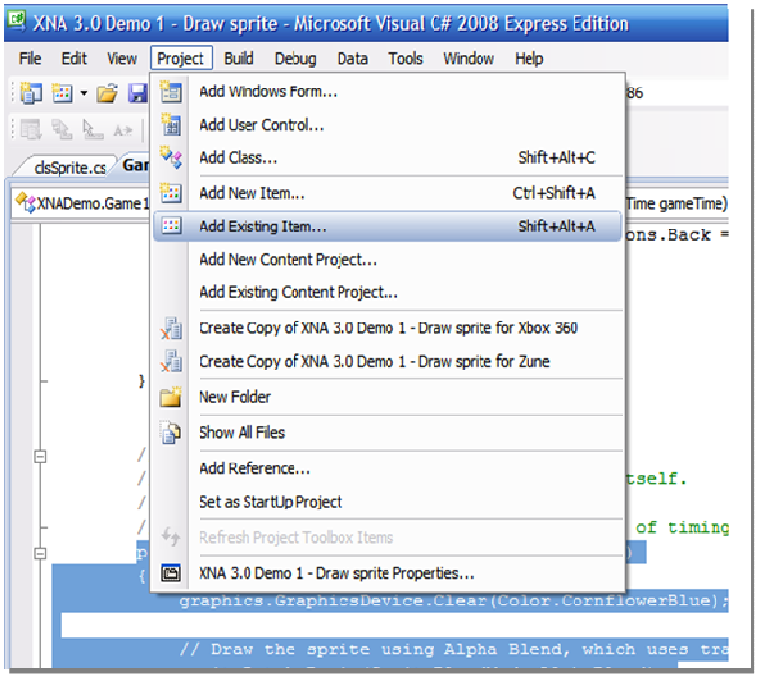
Tiếp theo một hàm để load tài nguyên cho game

```
protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            // Load a 2D texture sprite
            mySprite1 = new clsSprite(Content.Load<Texture2D>("ball"), new
Vector2(0f, 0f), new Vector2(64f, 64f));
}
```

mySpite1 sẽ đc load texture lên trên nó, file name là ball bạn add vào tài nguyên của game (project=>add=>existing items) file này có size 64x64 pixcel

```
            //câu lệnh trên có nghĩa texture bắt đầu lấy từ tọa độ 0,0 của
ball và kick cỡ là 64x64 (tức là cả file ball đó)
```



Tiếp là một hàm unload nhằm xóa các thành phần khỏi bộ nhớ khi close game:

```
protected override void
UnloadContent()
        {
            //  Free the
previously alocated
resources

mySprite1.texture.Dispose(
);

spriteBatch.Dispose();
        }
```

Hàm Update cứ giữ nguyên thế vì trong quá trình chạy, chả có quái j thay đổi cả.

Hàm đồ họa: đây là cái quan trọng nhất nó sẽ đưa Sprite lên bằng một texture ball tại vị trí nhất định:

```
protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

            spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
            mySprite1.Draw(spriteBatch);
            spriteBatch.End();

            base.Draw(gameTime);
              }
```

Chế độ tô kiểu AlphaBlend này sẽ cho phép chúng ta vẽ các bức ảnh có chỗ trống (một số chương trình như photóshop làm đc)

Sprite của chúng ta là mySpite1 sẽ lấy spriteBatch mà vẽ cho nó, trước một hàm Draw luôn có lệnh Begin(kiểu tô) và End();

Xong rùi, F5 nào J

Lưu ý file mẫu: có thể máy bạn ko nhận ra file ball.bmp vì khác đg dẫn, nếu thế cứ xoá file ball cũ đi, add mới nó vào ở mục content (kick phải chọn add existing item) tương tự cho các lession sau, nếu nó báo thiếu file, file not found thì cứ xoá hết rồi add lại

Bài 2 : Di chuyển Đối tượng trên màn hình

Các bạn có thể dùng tiếp project cũ để hoàn thành lession này.

Phần I: bổ sung cho lớp clsSprite:

Vật muốn di chuyển, ta thêm vận tốc cho nó: vector2 velocity

Thêm kick thước screen để check collided (va chạm): vector2 screenSize

Thuộc tính tọa độ điểm trung tâm của Sprite (đoạn sau dùng nó để viết thuật tóan va chạm)

Vector2 center

Thuộc tính bán kinh sprite(sau dùng nó để viết thuật tóan va chạm) float radius

```
        public Texture2D texture { get; set; } //  sprite texture, read-only
      property
        public Vector2 position { get; set; }  //  sprite position on screen
        public Vector2 size { get; set; }      //  sprite size in pixels
        public Vector2 velocity { get; set; }  //  sprite velocity
        private Vector2 screenSize { get; set; } //  screen size

        public Vector2 center{ get{ return position + (size/2);}  } //
sprite center
        public float radius { get { return size.X / 2; } } //  sprite radius
```

Thêm vào phần nhập dữ liệu:

```
public clsSprite (Texture2D newTexture, Vector2 newPosition, Vector2 newSize,
int ScreenWidth, int ScreenHeight){
            texture = newTexture;
            position = newPosition;
            size = newSize;
            screenSize = new Vector2(ScreenWidth, ScreenHeight);
        }
```

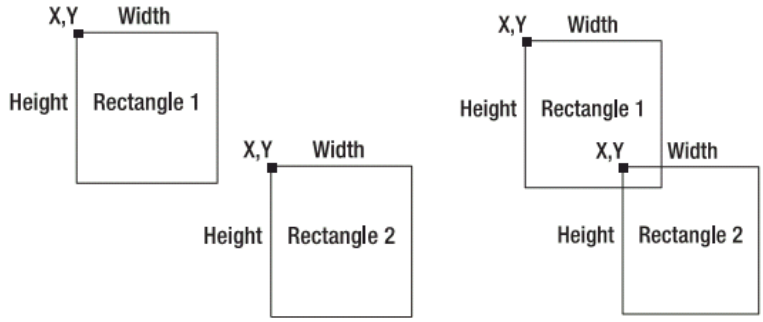
Chúng ta chỉ thêm một dữ liệu là chiều dài và rộng của screen.

Viết thêm một hàm bool trả về true nếu có trùng giữa 2 sprite clsSprite và false nếu ngược lại:

```
public bool Collides(clsSprite otherSprite)
        {
            // check if two sprites intersect
            return (this.position.X + this.size.X > otherSprite.position.X &&
                    this.position.X < otherSprite.position.X +
otherSprite.size.X &&
```

```
                    this.position.Y + this.size.Y > otherSprite.position.Y &&
                    this.position.Y < otherSprite.position.Y +
otherSprite.size.Y);
        }
```

Nó tính toán tọa độ và kick thước 2 sprite xem chúng có trùng nhau không, return trả về true nếu biểu thức trong () được thỏa mãn. Để biểu diễn các bạn có thể xem hình dưới đây:



Hàm này không cần thiết lắm vì chúng ta sử dụng nó trong lession3, chi viết để mang tính minh họa môt cách kiểm tra va chạm khác thôi, bạn không thêm nó vào cũng đc

Sau này chúng ta có cách khác để kiểm tra va chạm, đó là vẽ lên 1 hcn xung quanh mỗi sprite và kt va cham thông qua 2 hcn đó.

Thêm một hàm để kiểm tra va chạm biên của 2 sprite, so sánh khoảng cách giữa trung tâm 2 sprite và tổng bán kính của chúng: trả về true nếu biểu thức trong return() được thỏa mãn. Chung ta sử dụng hàm này để kiểm tra va chạm 2sprite trong quá trình chay game.

```
        public bool CircleCollides(clsSprite otherSprite)
        {    //  Check if two circle sprites collided
            return (Vector2.Distance(this.center, otherSprite.center) <
                this.radius + otherSprite.radius);
        }
```

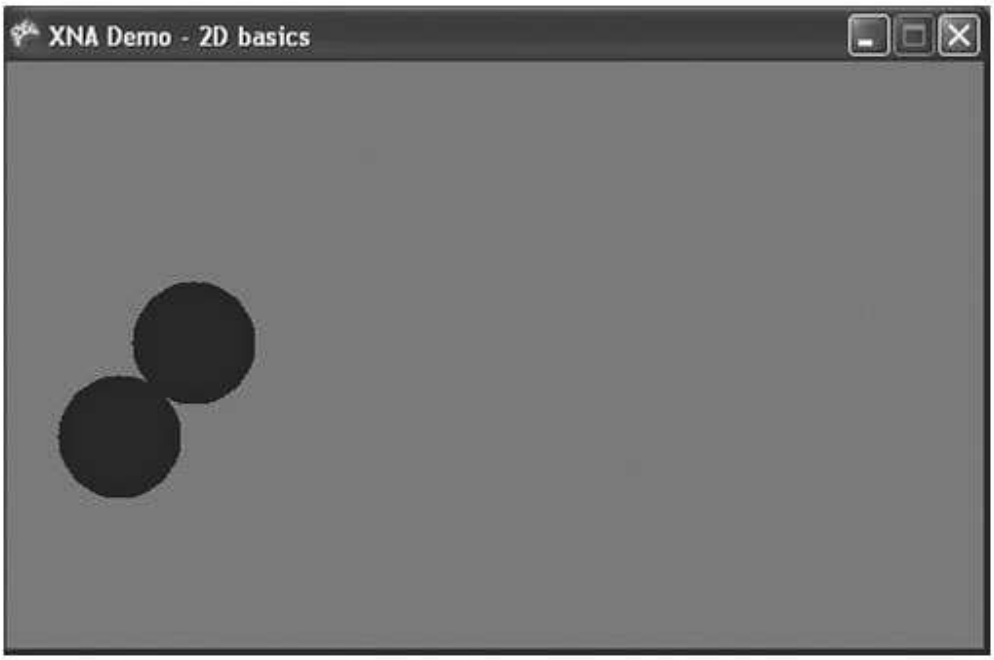


Figure 2-9. *The sprites now move and collide.*

Một hàm để kiểm tra có va chạm với biên hay không? Nếu có đảo giá trị velocity.X hoặc velocity.Y. nếu không thì sprite di chuyển (position tăng theo velocity của nó)

```
public void Move()
        {
            //  if we´ll move out of the screen, invert velocity

            //  checking right boundary
            if (position.X + size.X + velocity.X > screenSize.X)
                velocity = new Vector2(-velocity.X, velocity.Y);
            //  checking bottom boundary
            if (position.Y + size.Y + velocity.Y > screenSize.Y)
                velocity = new Vector2(velocity.X, -velocity.Y);
            //  checking left boundary
            if (position.X + velocity.X < 0)
                velocity = new Vector2(-velocity.X, velocity.Y);
            //  checking bottom boundary
            if (position.Y + velocity.Y < 0)
                velocity = new Vector2(velocity.X, -velocity.Y);

            //  since we adjusted the velocity, just add it to the current
position
            position += velocity;
        }
```

Hàm cuối cùng là hàm đồ họa thể hiện của sprite, rất quen thuộc:

```
public void Draw(SpriteBatch spriteBatch)
        {
            spriteBatch.Draw(texture, position, Color.White);
```

```
        }
```

Phan II: bổ sung cho Game1.cs

Chúng ta tạo ra 2 sprite lấy tên mySpite1 và mySprite2

```
        clsSprite mySprite1;
        clsSprite mySprite2;
```

Loadcontent cũng có 2sprite (load texture) khởi tạo giá trị tốc độ cho nó, theo 2 trục x và y:

```
protected override void LoadContent()
        {
            // Load a 2D texture sprite
            mySprite1 = new clsSprite(Content.Load<Texture2D>("ball"), new
Vector2(0f, 0f), new Vector2(64f, 64f),
                graphics.PreferredBackBufferWidth,
graphics.PreferredBackBufferHeight);
            mySprite2 = new clsSprite(Content.Load<Texture2D>("ball"), new
Vector2(218f, 118f), new Vector2(64f, 64f),
                graphics.PreferredBackBufferWidth,
graphics.PreferredBackBufferHeight);
            // Create a SpriteBatch to render the sprite
            spriteBatch = new SpriteBatch(graphics.GraphicsDevice);

            //  set the speed the sprites will move
            mySprite1.velocity = new Vector2(5, 5);
            mySprite2.velocity = new Vector2(3, -3);
        }
```

Bổ sung thêm hàm unLoad:

```
protected override void UnloadContent()
        {
            //  Free the previously alocated resources
            mySprite1.texture.Dispose();
            mySprite2.texture.Dispose();
            spriteBatch.Dispose();
        }
```

Bổ sung thuộc tính chiều dài và rộng của screen

```
            graphics.PreferredBackBufferWidth = 500;
            graphics.PreferredBackBufferHeight = 300;
```

2Sprite không chỉ di chuyển mà còn tương tác với nhau:Chúng ta kiểm tra va chạm trong hàm Update, bởi lẽ hàm Update sẽ kiểm tra game trong quá trình chạy để thay đổi nó, nó sẽ biết khi nào 2 sprite chạm nhau, chạm screenBound hoặc là không.

Nếu 2 sprite chạm nhau, chúng sẽ đảo tốc độ của nhau (vận tốc của sprite 1 sẽ chuyển qua sprite2 và ngược lại)

Diều này nghiệm đúng ĐL bảo toàn động lượng trong vật lý J

```
protected override void Update(GameTime gameTime)
        {
            // Move the sprites
            mySprite1.Move();
            mySprite2.Move();

            if (mySprite1.CircleCollides(mySprite2))
            {
                Vector2 tempVelocity = mySprite1.velocity;
                mySprite1.velocity = mySprite2.velocity;
                mySprite2.velocity = tempVelocity;
            }

            base.Update(gameTime);
        }
```

Cơ bản là xong thêm hàm đồ họa nữa, nó sẽ vẽ sprite1 và sprite2 lên screen:

```
protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);

            // TODO: Add your drawing code here
            spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
            mySprite1.Draw(spriteBatch);
            mySprite2.Draw(spriteBatch);
            spriteBatch.End();

            base.Draw(gameTime);
        }
```

Xong rùi, F5 nào J .

Bài 3: INPUT cho OBJECT bằng MOUSE và KEYBOARD

Bài3 này chúng ta sử dụng thuật tóan KT va chạm khác, đã nói qua ở lession 2

Bạn có thể sử dụng project của lession trước + một và sửa đổi ở đây:

Phần I: từ lớp clsSprite:

1.chung ta không cần thuộc tính center và radius nữa.

2.Sử dụng hàm va chạm:

```
public bool Collides(clsSprite otherSprite)
        {
            // check if two sprites intersect
            if (this.position.X + this.size.X > otherSprite.position.X &&
                    this.position.X < otherSprite.position.X +
otherSprite.size.X &&
                    this.position.Y + this.size.Y > otherSprite.position.Y &&
                    this.position.Y < otherSprite.position.Y +
otherSprite.size.Y)
                return true;
            else
                return false;
        }
```

Hàm này trả về false nếu 2 sprite (giống lession 2) không va chạm và ngược lại sẽ trả về true.

Hàm đã nói ở lession2.

Các hàm Move() và Draw() không thay đổi.

Phần II : từ lớp Game1.cs

1.khai báo: sau class Game1:`Microsoft.Xna.Framework.Game;`

Chúng ta cần 2 sprite như lession trước.

Mục Load content tải đồ họa cho các sprite và thiết lập vận tốc cho sprite1, còn sprite2 không cần thiết lập tốc độ (velocity) vì chúng ta sẽ điều khiển nó từ mouse hoặc keyboard.

Khai báo:

```
public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;


        //  Sprite objects
        clsSprite mySprite1;
        clsSprite mySprite2;
```

Hàm LoadContent:

```
protected override void LoadContent()
        {
            // Create a new SpriteBatch, which can be used to draw textures.
            spriteBatch = new SpriteBatch(GraphicsDevice);

            mySprite1 = new clsSprite(Content.Load<Texture2D>("ball"), new
Vector2(0f, 0f), new Vector2(64f, 64f),
                graphics.PreferredBackBufferWidth,
graphics.PreferredBackBufferHeight);
            mySprite2 = new clsSprite(Content.Load<Texture2D>("ball"), new
Vector2(218f, 118f), new Vector2(64f, 64f),
                graphics.PreferredBackBufferWidth,
graphics.PreferredBackBufferHeight);

            //  set the speed the sprites will move
            mySprite1.velocity = new Vector2(5, 5);
        }
```

Hàm checkKeyboard() nằm trong hàm Update() sẽ có nhiệm vụ kiểm tra phím đc nhấn trong thgời gian chạy game. Nói chung cái j cần kt trong khi run game đều put trong hàm Update() này. Chúng ta ĐK sprite2

```
protected void checkKeyboard()
        {
         KeyboardState keyboardState = Keyboard.GetState();
```

Khởi tạo bàn phím

```
            if (keyboardState.IsKeyDown(Keys.Up))
                mySprite2.position += new Vector2(0, -5);
```

Vị trí thay đổi khi nhấn xuống (press) với tọa độ thay đổi là 5, tùy theo hứong phím mũi tên mà tọa độ thay đổi thích hợp

bạn cần nhớ trục tọa độ của XNA là hệ Oxy với tâm O nằm trên góc trên cùng bên trái bạn, Ox // mặt đất và Oy Vuông góc mặt đất.

```
            if (keyboardState.IsKeyDown(Keys.Down))
                mySprite2.position += new Vector2(0, 5);
            if (keyboardState.IsKeyDown(Keys.Left))
                mySprite2.position += new Vector2(-5, 0);
            if (keyboardState.IsKeyDown(Keys.Right))
                mySprite2.position += new Vector2(5, 0);


            if (mySprite1.Collides(mySprite2))
```

Hàm mySpite1.Collides sẽ xác định va chạm với sprite2 trả về true nếu có.

```
            {
                mySprite1.velocity *= -1;
```

Đảo chiều chuyển động của sprite1

```
            }
```

```
        }
          }
```

Tiếp là một hàm điều khiển sprite2 bằng mouse, khi chạy 1 hoặc 2 điều đc nhưng tốt nhất bạn chọn lấy 1 trong 2 hàm cho đỡ lộn xộn.

```
protected void checkMouse()
{


//  Hàm lệnh sẽ giúp sprite2 di chuyển đến tọa độ của mouse
            if (mySprite2.position.X < Mouse.GetState().X)
                mySprite2.position += new Vector2(5, 0);
            if (mySprite2.position.X > Mouse.GetState().X)
                mySprite2.position += new Vector2(-5, 0);
            if (mySprite2.position.Y < Mouse.GetState().Y)
                mySprite2.position += new Vector2(0, 5);
            if (mySprite2.position.Y > Mouse.GetState().Y)
                mySprite2.position += new Vector2(0, -5);
```

Vector2 là 1 cặp biến nghĩa là khi sử dụng vector ta cũng phải dùng với 1 cặp giá trị.

```
if (mySprite1.Collides(mySprite2))
            {
                mySprite1.velocity *= -1;

            }

}
```

Trong file mẫu có thêm hàm cho tay cầm của Xbox rung, nhưng trong bài viết mình remove rồi, vì hiện tại chưa có con Xbox nào để test cả J .

Xong rồi đấy, F5 và thử con chuột hay bàn phím đi nào .

Lưu ý , file mẫu có 2 code cho việc control bằng mouse hay keyboard, mouse đang ở phần ghi chú (tức là có // trước câu) muốn test các bạn enable đoạn code cho mouse đó là đc, (khi đó nên disable code cho keyboard cho dễ thấy)

Bài 4 SỬ DỤNG ÂM THANH TRONG XNA

Tài nguyên: texture ball. 2 sound, một cái làm backmusic, cái kia làm effectmusic

Bất kể game đều cần phải có chút âm thanh để gây hưng phấn cho gamer. Có 2 loại phổ biến: nhac nền và nhạc hiệu ứng

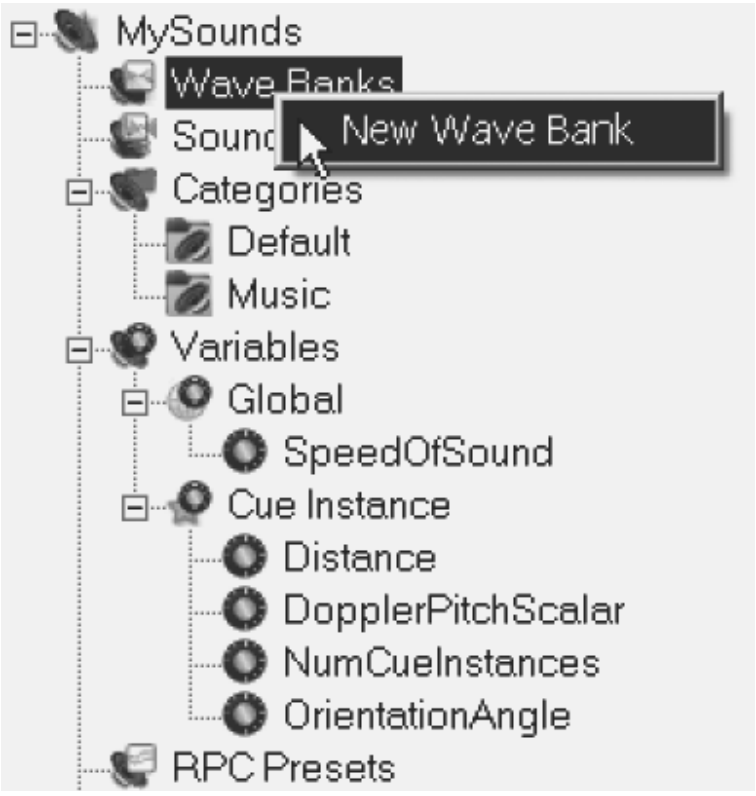
-nhạc nền chạy khi bắt dầu, kết thúc hoăc cả khi đang play.

-nhạc hiệu ứng sẽ đc chơi tùy trường hợp: va chạm, ăn điểm, win...

Phần I: Khởi tạo những thứ cần thiết

Chúng ta chuẩn bị cho âm thanh từ một hệ thống quản lý âm thanh của XNA đó là phatform Audio Creation Tools.các bước như sau:

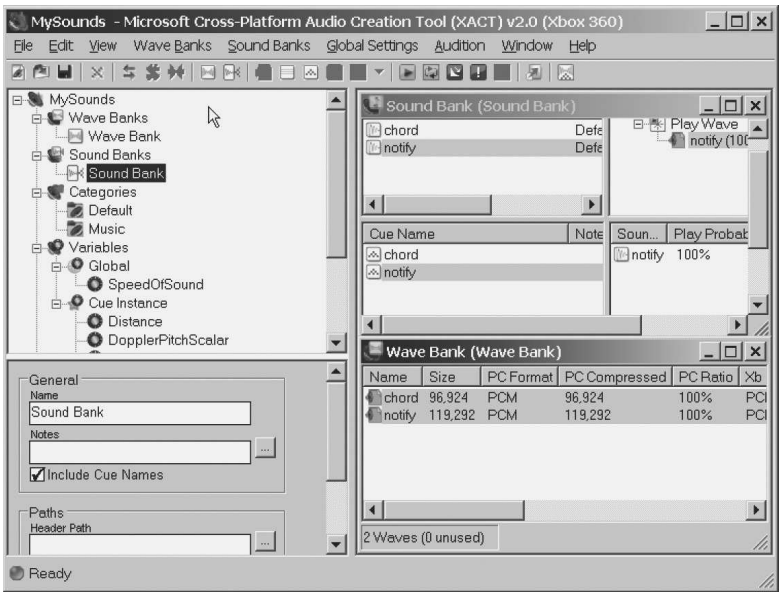
1. bắt đầu XACT bằng cáchchọn Start ➢ Programs ➢ Microsoft XNA Game Studio 3.0 ➢ Tools ➢ Cross-Platform Audio Creation Tool (XACT).
2. trong bang XACT window, chọn File ➢ New Project để tạo mới audio project, và lưu lại với tên MySounds.
3. nhìn phía bên trái cửa sổ, MySounds sẽ xuất hiện với nhiều thành phần khác. Kick chuột phải vào Wave Bank và lựa chọn New Wave Bank trong cửa sổ hiện ra
Như hình:



Kick chuột phải vào sound banks chọn new sound bank

Vô windows => tiltle hórizon

Sau đó drag và drop nhạc từ wave bank vào cue box, lưu nó lại



Sau khi thực hiện xong bạn sẽ có file MySounds.xap đặt nó trong folder chứa project

Lưu ý, trong file mẫu, do đường dẫn file khác nhau (ở thực tế máy bạn và ở project) nên chắc máy bạn ko chay đc, tốt nhất là tự tạo 1 file XAP mới rồi kick phải content ở cưa sổ solution chọn add existing item rồi add nó vào

Phan II Thêm mã lệnh

Bạn sử dụng lại dự án cũ.

Thêm vào mục khai báo sau lớp class Game1:...

Các yếu tố của audio:

```
        AudioEngine audioEngine;
        WaveBank waveBank;
        SoundBank soundBank;
        Cue myLoopingSound = null;
```

Chúng ta sử dụng Hàm Initiallize() để thiết lập âm thanh lặp (dùng làm nhạc nền):

```
protected override void Initialize()
        {
            audioEngine = new AudioEngine(@"Content\MySounds.xgs");

            //  Đường dẫn đến file hệ thống
            //   BẠN CẦN add file MySounds.xap vào
            waveBank = new WaveBank(audioEngine, @"Content\Wave Bank.xwb");
            soundBank = new SoundBank(audioEngine, @"Content\Sound
Bank.xsb");

            myLoopingSound = soundBank.GetCue("notify");
```

```
            myLoopingSound.Play();

            base.Initialize();
        }
```

Âm thanh lặp `notify` Sẽ coi là nhạc nền vì nó lặp lại và không bao giờ ngừng cho đến khi end. Chúng ta đặt nó trong hàm thiết lập.

Bổ sung nhạc hiệu ứng:

```
if (mySprite1.Collides(mySprite2))
            {
                mySprite1.velocity *= -1;
                soundBank.PlayCue("chord");
            }
```

Khi nào sprite1 va chạm với sprite2 chúng ta sẽ chơi nhạc hiệu ứng nó được gọi khi sự kiện này xảy ra, mục đích là báo cho chúng ta sự va chạm này bằng thính giác, tất nhiên khi chơi game, nhạc hiệu ứng cũng là cách giúp chúng ta kiểm soát cuộc chơi của mình tốt hơn và sống động hơn

Cho nhạc chạy và dừng nhạc nền, nhạc hiệu ứng không cần vì nó chỉ đc gọi khi có sự kiện đặc biệt (có va chạm chẳng hạn) nhưng môt số người không cần nhạc nền hay vì lý do nào đó, họ muốn tắt nhạc nền đi, đặt code dưới đây và hàm Update() nó sẽ giúp tắt hay mở nhạc nền `myLoopingSound` Khi bạn nhấn phím Enter.

```
if (keyboardState.IsKeyDown(Keys.Enter))            {
                if (myLoopingSound.IsPaused)
                    myLoopingSound.Resume();
                else
                    myLoopingSound.Pause();
            }

            base.Update(gameTime);
        }
```

Đó là những thứ rất cơ bản để bạn thêm âm thanh vào game của bạn trong XNA, hết bài 4,

Have Fun!

BÀI 5: Lập trình Hướng đối tượng Cơ bản của C# Trong XNA

Tài nguyên: 1 background, 1 image gộp chung trong đó 2 texture của Ship và Meteor

Sound: 2 tiếng bùm (effect-nhạc hiệu ứng),1 tèn tén ten (backmusic-nhac nền)

Font: 1 font nào đó, chúng ta sẽ cần đưa text lên game

Nếu đã rành C# xin mời qua luôn phần II J

Phần I:Tính đóng gói và Tính đa hình của OOP

Câu hỏi đặt ra: tại sao LT hướng đối tượng hay đc sử dụng khi viết game? Game đầu tay của mình viết bằng flash và nó thực sự rất dễ, để tạo object chỉ cần drag and drop, viết mã lệnh cho từng thằng bằng phím F9 với các thuộc tính sẵn có (góc, toạ độ, độ mờ đục...) và đủ để biến nó thành sprite thứ thiệt. KT va chạm cực kỳ đơn giản chỉ với 1lệnh hitTest. Bạn có thể tượng tưởng một thằng nhóc 17 tuổi với game đầu tay của nó sẽ háo hức ntn?, giờ nhìn lại mình thấy tất cả chỉ là khởi đầu, cái đc chỉ là những giải pháp lập trình tự tìm tòi đc . nhưng Khi học XNA, C# khó hơn flash (dùng Action Script) cả tỉ lần, ko thể viết code một cách cẩu thả, riêng việc dựng object lên screen cũng là vấn đề. Tuy vậy C# là ngôn ngữ chặt chẽ, sự khó khăn khi lập trình ban đầu đưa lại chúng ta sự cẩn thận + Hệ thống quản lý tốt của XNA khiến bạn phải cẩn thận từng câu lệnh và không thể chữa cháy một hàm mà ta ko biết cách viết bằng hoạt hình chả hạn, mình nhận ra rằng 1 game nhỏ viết bằng flash là đủ những một game kha khá, một dự án lớn thì không thể, lập trình OOP có những đặc tính mà LT truyền thống (theo tuyến tính chạy từ đầu về cuối hay lập trình hàm, thủ tục, vd: pascal) không thể có đc, mạnh nhất là tính đóng gói, đa hình và kế thừa.

Chác nhiều bạn đã biết về tính đa hình? Các bạn đã học qua c# rồi mà, mình hi vọng vậy J

Từ khi bắt đầu tut, chúng ta đã dùng đến thằng OOP này bởi lẽ C# là NNLT hướng đối tượng hoàn toàn, tuy nhiên lession chúng ta sẽ đi sâu vào vấn đề nêu ở tiêu đề K , Phần II sẽ là một project game đầu tiên của tut, cũng đơn giản thôi J .

OOP là một vấn đề khó, những thứ trong tut này là những EXP đúc kết qua quá trình LT và làm game hoàn toàn mang tính cá nhân chắc không thể hoàn toàn chính xác đc.

Đóng gói (**Encapsulation)**: Class mà bạn viết ra cho 1 con monster còn gọi là tính đóng gói của OOP. tính đóng gói thể hiện tất cả khả năng của object trong một class... bao gồm cả việc kiểm soát nó bằng cách đánh dấu cho thuộc tính và methods(phương thức) là public (tự do truy cập), private (hạn chế hơn)..., hăcọ có thể hiểu rằng tính đóng gói thể hiện ở cách sử dụng từ methods (phương thức) nằm trong class của một object nào đó làm chương trình trở lên ngắn gọn và rành mạch, ví dụ như bạn có methods RUN(int speed) cho con sleketon, bạn muốn cho nó speed là 500 : sleketon.RUN(500);...

Đa hình (**Polymorphism)**: Một class là gói lại toàn bộ khả năng mà object đc khai báo kiểu class đó có đc. Nhưng OOP cho phép chúng ta tạo ra nhiều objêct hoạt động dựa trên class đó

một cách tương tự nhau, nếu có muốn có sự khác nhau trong cách hoạt động của từng object chúng ta sẽ sử dụng tính kế thừa (lession sau)

lấy ví dụ nhé, 1 game có 100 con monster, bạn sẽ ngồi code cho 100 con monster? Thật vớ vẩn đúng không. Nhưng trong số 100 con monster giống nhau, bạn sẽ chỉ viết mã lệnh cho 1 con thôi và muốn tạo ra 100 con khác sẽ hoạt động theo mã lệnh đó.

C# là NNLT hướng đối tượng hoàn toàn, do đó bạn khi bạn viết code thì bạn cũng đã khởi tạo cho tính đa hình của mình. những con monster cùng class sẽ hoạt động như nhau trong cùng điều kiện. Sử dụng tính đa hình tuy tiện lợi nhưng cũng cần cẩn thận tối đa, bạn nên chắc chắn có thể kiểm soát 100 con monster khi chung nó đc taọ ra. Lession sau chúng ta sẽ sử dụng tính thừa kế để nâng cấp dự án. Nó cũng cải thiện sức mạnh của tính đa hình một cách đáng kể

Trong XNA, một thứ trên screen nếu tạo ra một cách động (dùng tính đa hình-mã lệnh) hay cho dù là không đều gọi là 1 component và nó sẽ kế thừa từ lớp
`Microsoft.Xna.Framework.DrawableGameComponent` Hoặc
`Microsoft.Xna.Framework.GameComponent.`

Tính đa hình thể hiện trong lớp Game1.cs: Lớp của Object nào chỉ chứa những khả năng của Object đó. Bằng câu lệnh sau đặt nó trong hàm `LoadContent()` hay hàm mà đc gọi ít nhất 1 lần trong Game1.cs:

```
Services.AddService(typeof(SpriteBatch), spriteBatch);
```

Nó sẽ thêm component (thành phần) vào chương trình một cách động.

Để add component, chúng ta dùng kỹ thuật sau:

Int OBJECTCOUNT = 10

```
for (int i = 0; i < OBJECTCOUNT; i++)
            {
                Components.Add(new OBJECT(this, ref OBJECTTexture));
            }
```

Vòng lặp for trên tạo ra 10 (số lượng) các `OBJECT` trên màn hình.

Lệnh này thường là đóng vai trò begin cho một game, nó cũng thường chỉ gọi 1 lần, trừ khi người chơi gameover và muốn restart.

Để xoá các component đc tạo ra theo cách động chúng ta sử dụng kỹ thuật sau:

```
private void RemoveAllOBJECT()
        {
            for (int i = 0; i < Components.Count; i++)
            {
                if (Components[i] is OBJECT)
                {
                    Components.RemoveAt(i);
                    i--;
                }
```

```
            }
        }
```

Các cômponent đó đc tập hợp trong một mảng có tên `Components` Mảng này do XNA tự tạo. Lệnh `Components.Count` trả về số lượng component trong chương trình, vòng lặp for sẽ quét toàn bộ các component đó và KT xem thằng nào là OBJECT, nếu là nó thì Remove nó đi, sai đó hạ i xuống để KT thằng tiếp theo.

Chúng ta còn muốn tạo ra OBJECT tuỳ theo ĐK , hoàn cảnh của game như sau một thời gian xác định, khi boss kêu thêm đệ của nó,.... cấu trúc như sau:

```
private void CheckforNewOBJECT()
        {             if (Điều kiện để có thêm OBJECT)
            {

                  Components.Add(new Meteor(this, ref meteorTexture));
                  OBJECTCount++;
            }
        }
```

Trong kỹ thuật trên chúng ta cần:

Thêm Câu lệnh để đảm bảo ĐK OBJECT thứ 2 vẫn có thể ra đời

Biến OBJECTcount sẽ đếm số lượng OBJECT đã đc tạo giúp chúng ta có thể kiểm soát nó.

Vẽ Texture cho component: khác với trước đây, texture đc quy định trong chính class của OBJECTđiều này chúng ta sẽ đi sâu khi làm 1 project cụ thể, thế lên, trong Game1.cs chúng ta sử dụng hàm đồ hoạ rất đơn giản.

```
spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
            base.Draw(gameTime);
            spriteBatch.End();
```

Kết thúc phần I, chúng ta sẽ làm project đầu tiên ngay bây giờ

Phần II: DỰ ÁN ĐẦU TIÊN

Trước khi bắt đầu dự án, ý tưởng là khâu rất quan trọng, bạn có thể bỏ vài tuấn viết game chả nhẽ không bỏ đc vài ngày viết ý tưởng.

Ý tưởng của dự án: Chúng ta Điều khiển phi thuyền bằng phím mũi tên tránh các vật cản trong không gian là các thiên thạch để tránh bị nó phá huỷ. Ship và Mêteor là những component của game và chúng đc tạo ra một cách động. Sau một khaỏng thời gian nhất định sẽ có thêm Meteor đc tạo ra để tăng độ khó cho game, khi phi thuyền bị phá huỷ, game sẽ đc restart lại. Chúng ta tạo ra text field để hiện số lượng Meteor

AI: trí thông minh nhân tạo.

Mặc dù project sắp viết dưới đây có AI kiểu "Random" tức là nó ngẫu nhiên không phụ thuộc player, nhưng bạn cũng nên biết chút ít về một AI tốt:

-Dễ bị đánh bại: Gamer nào cũng muốn win, nếu họ ko win đc, họ sẽ ko đời nào chơi game đó.

-không dễ bị đánh bại: Đối thủ quá yếu làm gamer thấy chán và họ đi tìm một game khó hơn.

-Khả năng thay đổi: Nó có thể từ dễ trở thành khó hơn khi người chơi lên tay và giữ họ ngồi xuống lâu hơn để chinh phục các mức độ mới

-Khả năng sống động: AI cần có tính logic, có thể dự đoán đc giống như là một con người đnag chơi cùng bạn vậy (di chuyển, tấn công...) giúp người chơi có thể phán đoán và chiến thắng trong game.

Chúng ta áp dụng khả năng tăng độ khó cho trò chơi trong project này.

Chúng ta tiến hành lấy texture cho tư`ng object cụ thể trong một texture chung, cái này sẽ làm gọn lại chương trình tránh import nhiều.

Bắt tay vào công việc thôi:

1.Đối tượng đàu tiên là phi thuyền (Ship) kế thừa từ
`Microsoft.Xna.Framework.DrawableGameComponent`

Và nó sử dụng:

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
```

Như thường lệ đầu tiên là khai báo biến:Các thành phần không thể thiếu:Texture,Position, Chiều dài, rộng của Ship (hằng số) `SpriteBatch,` Những thứ bổ sung là Hình chữ nhật bao quanh Ship để kiểm tra va chạm (spriteRectangle) và Hình chữ nhật là khung của màn hình thể hiện. Thuộc tính vận tốc không cần nữa, chúng ta thay đổiví trị thông qua position. HẰng số chiều dài và rộng của ship để lấy chính xác texture của ship trên file texture (gồm nhiều texture của các Object khác nhau)

```
        protected Texture2D texture;
        protected Rectangle spriteRectangle;
        protected Vector2 position;
        protected SpriteBatch sBatch;

        protected const int SHIPWIDTH = 30;
        protected const int SHIPHEIGHT = 30;

        protected Rectangle screenBounds;
```

screenBounds chủ yếu là giúp việc xác định va chạm với màn hình dễ dàng hơn thông qua các cạnh của hcn.

Tiếp theo là hàm nhập dự liệu: Chúng ta nhập cho nó texture và game cái mà nó sẽ chạy trên

Vị trí chúng ta sẽ thêm vào khi tạo ra nó (trong hàm khác), một lệnh lấy dịch vụ (một trong số đó là cho phép sử dụng ship về tính đa hình) cho sBatch, cái thể hiện cho Ship

```
public Ship(Game game, ref Texture2D theTexture)
            : base(game)
        {
            texture = theTexture;
            position = new Vector2();

      sBatch =(SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));

      spriteRectangle = new Rectangle(31, 83, SHIPWIDTH,SHIPHEIGHT);


            screenBounds = new Rectangle(0,0,
                Game.Window.ClientBounds.Width,
                Game.Window.ClientBounds.Height);

        }
```

Hàm đặt phi thuyền vào vị trí khi bắt đầu:

```
public void PutinStartPosition()
        {
            position.X = screenBounds.Width / 2;
            position.Y = screenBounds.Height - SHIPHEIGHT;
        }
```

Thêm mã lệnh để tàu có thể điều kiển , tất nhiên nó đặt trong hàm Update nơi luôn KT điều kiện của game:

```
KeyboardState keyboard = Keyboard.GetState();

//Khởi tạo bàn phím

if (keyboard.IsKeyDown(Keys.Up))
            {
                position.Y -= 3;
            }
            if (keyboard.IsKeyDown(Keys.Down))
            {
                position.Y += 3;
            }
            if (keyboard.IsKeyDown(Keys.Left))
            {
                position.X -= 3;
            }
            if (keyboard.IsKeyDown(Keys.Right))
            {
                position.X += 3;
            }
```

Một mã lệnh nữa để đảm bảo cho Ship ko đi quá xa, luôn nằm trên màn hình: màn hình đc gọi thông qua một hình chữ nhật bao quanh nó.

```
if (position.X < screenBounds.Left)
            {
                position.X = screenBounds.Left;
            }
            if (position.X > screenBounds.Width - SHIPWIDTH)
            {
                position.X = screenBounds.Width - SHIPWIDTH;
            }
            if (position.Y < screenBounds.Top)
            {
                position.Y = screenBounds.Top;
            }
            if (position.Y > screenBounds.Height - SHIPHEIGHT)
            {
                position.Y = screenBounds.Height - SHIPHEIGHT;
            }
```

Để kiểm tra Ship có va chạm với Meteor không, chúng ta đi KT 2 hình chữ nhật bao quanh 2 object đó, muốn vậy cần biết HCN của Ship là cái j đã, hàm sau sẽ trả về giá trị đó:

```
public Rectangle GetBounds()
        {
            return new Rectangle((int)position.X, (int)position.Y,
                SHIPWIDTH, SHIPHEIGHT);
        }
```

Hàm đồ hoạ: cái thêm vào ở đây là XNA sẽ vẽ Ship theo một hình chữ nhật chính là thằng spriteRectangle, HCN bao phủ lấy phần texture chung (gồm cả Ship và Meteor) ở toạ độ 31;83 và bao phủ hoàn toàn hình cái phi thuyền trên texture , XNA cắt cái hìnhcn này ra mà biến nó là texture của Ship. cái này thêm ở lệnh Draw:

```
public override void Draw(GameTime gameTime)
        {
            // Vẽ con tàu
            sBatch.Draw(texture, position, spriteRectangle, Color.White);

            base.Draw(gameTime);
        }
```

Xong thằng Ship, tiếp theo là Meteor

2>Class Meteor

Meteor giống với ship về using và thừa kế.

Meteor tự nó di chuyển tự do chứ không phải control, do đó thêm thuộc tính vận tốc và một số ngẫu nhiên cho nó:

```
public class Meteor : Microsoft.Xna.Framework.DrawableGameComponent
    {
        protected Texture2D texture;
```

```
        protected Rectangle spriteRectangle;
        protected Vector2 position;
        protected int Yspeed;
        protected int Xspeed;
        protected Random random;
        protected SpriteBatch sBatch;


        protected const int METEORWIDTH = 45;
        protected const int METEORHEIGHT = 45;
```

Đây là kỹ thuật tạo ra một số ngâu nhiên:

random = new Random(this.GetHashCode());

Sau đó, một hàm để Meteor xuất hiện phía trên screen một cách ngẫu nhiên và có vận tốc ngẫu nhiên:

```
protected void PutinStartPosition()
        {
            position.X = random.Next(Game.Window.ClientBounds.Width -
METEORWIDTH);
            position.Y = 0;
            Yspeed = 1 + random.Next(9);
            Xspeed = random.Next(3) - 1;
        }
```

Lưu ý một tí `random.Next(a) sẽ trả về số random trong(0;a)`

Hàm nhập dữ liệu cho Meteor cũng tương tự với Ship:

```
public Meteor(Game game, ref Texture2D theTexture)
            : base(game)
        {
            texture = theTexture;
            position = new Vector2();

            sBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));


            spriteRectangle = new Rectangle(20, 16, METEORWIDTH,
METEORHEIGHT);

            random = new Random(this.GetHashCode());
            PutinStartPosition();
        }
```

Hàm Update giúp Meteor di chuyển mà tự restart lại khi vượt biên :

```
public override void Update(GameTime gameTime)
        {
            // Luôn để Meteor trong screen
            if ((position.Y >= Game.Window.ClientBounds.Height) ||
```

```
                (position.X >= Game.Window.ClientBounds.Width) || (position.X
<= 0))
            {
                PutinStartPosition();
            }

            // di chuyển meteor
            position.Y += Yspeed;
            position.X += Xspeed;

            base.Update(gameTime);
        }
```

Một Hàm tạo ra một hình cn xung quanh Mêteor để kiểm tra va chạm với Ship thông qua KT va chạm với hcn bao quanh Ship, trả về True hoặc false:

```
public bool CheckCollision(Rectangle rect)
        {
            Rectangle spriterect = new Rectangle((int)position.X,
(int)position.Y,
                METEORWIDTH, METEORHEIGHT);
            return spriterect.Intersects(rect);
    }
```

Phương thức Intersects() Thực sự rất hiệu quả và đơn giản khi KT xem 2 HCN có trùng nhau không.

Cuối cùng là Hàm đồ hoạ, nó cũng như của Ship vậy:

```
public override void Draw(GameTime gameTime)
        {
            // Draw the meteor
            sBatch.Draw(texture, position, spriteRectangle, Color.White);

            base.Draw(gameTime);
        }
```

3. Game1.cs – Nơi mọi thứ hoạt động

Sử dụng:

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
```

Kế thừa:

```
public class Game1 : Microsoft.Xna.Framework.Game
```

Khai báo:

```
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
```

```
        private Texture2D backgroundTexture, meteorTexture;

        // âm thanh
        private SoundEffect explosion;
        private SoundEffect newMeteor;
        private Song backMusic;

        // Thành phần object
        private Ship player;
        private int lastTickCount;
        private KeyboardState keyboard;
        private SpriteFont gameFont;
        private int rockCount;
        private const int STARTMETEORCOUNT = 10;
        private const int ADDMETEORTIME = 5000;
```

Texture gồm hình nền và hình của các object.

Âm thanh chúng ta add trực tiếp như texture, song sử dụng như nhạc nền, SoundEffect chắc các bạn hiểu rồi. Khai báo font vì chúng ta sẽ ghi số lượng Meteor hiện tại lên màn hình cho gamer biết mà tránh. Một biến đếm số lượng Meteor giúp kiểm soát chúng, hằng `STARTMETEORCOUNT = 10;` và `ADDMETEORTIME = 5000;` Có tác dụng đưa ra số lượng Meteor tối thiểu khi begin game và Khoảng Thời gian chúng ta tạo thêm meteor để tăng độ khó cho trò chơi

Hàm nhập dữ liệu: nó chả có j ngoài những thứ bạn đã biết:

```
public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
        }
```

Hàm thiết lập bổ sung câu lệnh sau để ghi tiêu đề cho game:

```
 protected override void Initialize()
        {
            base.Initialize();
            Window.Title = "RockRain";
        }
```

Hàm LoadContent, load đồ hoạ sử dung (graphic, texture) âm thanh và SpriteBatch:

```
protected override void LoadContent()
        {

            spriteBatch = new SpriteBatch(GraphicsDevice);
            // thêm vào dịch vụ spritebatch service
            Services.AddService(typeof(SpriteBatch), spriteBatch);

            // Load tất cả textures gồm ảnh nền và objects
            backgroundTexture = Content.Load<Texture2D>("SpaceBackground");
            meteorTexture = Content.Load<Texture2D>("RockRain");
            // Load game font
```

```
            gameFont = Content.Load<SpriteFont>("font");
            // Load các sound trong game
            explosion = Content.Load<SoundEffect>("explosion");
            newMeteor = Content.Load<SoundEffect>("newmeteor");
            backMusic = Content.Load<Song>("backMusic");

            // Chơi nhạc nền
            MediaPlayer.Play(backMusic);
        }
```

MediaPlayer.Play(backMusic); Sẽ làm chạy nhạc nền cho đến khi end game. Hoặc khi bài hát end;

1.Đến khâu tạo ra các meteor:

```
for (int i = 0; i < STARTMETEORCOUNT; i++)
            {
                Components.Add(new Meteor(this, ref meteorTexture));
            }
```

Meteor(this, ref meteorTexture)); Nếu các thắc mắc về các tham số thís và meteorTexture hãy coi lại phần nhập dữ liệu của lớp Meteor

2.Để tạo ra ship, chúng ta ko cần đến for bởi chỉ có 1 ship thôi mà, bởi vậy định nghĩa kiểu truyền thống và mảng component của XNA sẽ add nó vào:

```
player = new Ship(this, ref meteorTexture);
                Components.Add(player);
```

3.Sau khi add thì đặt nó cho đúng chỗ nữa, các bạn có nhớ hàm nào đã viết đảm nhận việc này ko?

4.Ngoài ra ta thiết lập thời gian `lastTickCount` là thời điểm hiện tại của hệ thống lúc bắt đầu nữa và số lượng meteor(rockcount) sẽ bằng số Meteor khởi đầu `STARTMETEORCOUNT`:

```
lastTickCount = System.Environment.TickCount;
            rockCount = STARTMETEORCOUNT;
```

Giờ hãy xây dựng một các lôgic chúng để có 1 hàm start hay dùng restart:

```
private void Start()
        {
            // Add the meteors
            for (int i = 0; i < STARTMETEORCOUNT; i++)
            {
                Components.Add(new Meteor(this, ref meteorTexture));
            }

                        if (player == null)
            {
                player = new Ship(this, ref meteorTexture);
                Components.Add(player);
            }
//tạo ra player(Ship) nếu chưa có
            player.PutinStartPosition();
```

```
//đặt vào vị trí.
            lastTickCount = System.Environment.TickCount;
            rockCount = STARTMETEORCOUNT;
        }
```

Viết hàm để biết khi nào nên thêm meteor:

```
private void CheckforNewMeteor()
        {
            // Add a rock each ADDMETEORTIME
            if ((System.Environment.TickCount - lastTickCount) >
ADDMETEORTIME)
            {
                lastTickCount = System.Environment.TickCount;
                Components.Add(new Meteor(this, ref meteorTexture));
                newMeteor.Play();
                rockCount++;
            }
        }
```

System.Environment.TickCount trả về thời điểm hiện tại của hệ thống, System.Environment.TickCount – lastTickCount trả về khoảng time từ lúc bắt đầu (bạn KT xem lại có fải thế k?) nếu nó > ADDMETEORTIME, thì **tiếp tục nâng thời điểm lastTickCount = thời điểm hiện tại để sau này trong hàm Update sẽ tiếp tục kiểm tra thời gian chạy game mà add thêm meteor sau khoảng time ADDMETEORTIME;** sau đó thì đơn giản rồi, add thêm meteor , newMeteor.Play(); và tăng biến rockCount++;

Điều j xay ra nếu Ship va vào meteor? Bùm sau đó game đc restart, chúng ta tận dụng hàm Start để restart luôn còn các meteor phải trở về với 0 của nó bằng hàm sau:

```
private void RemoveAllMeteors()
        {
            for (int i = 0; i < Components.Count; i++)
            {
                if (Components[i] is Meteor)
                {
                    Components.RemoveAt(i);
                    i--;
                }
            }
        }
```

Components là mảng chứa các component (thành phần) của XNA nhắc lại nếu các bạn wen.

Tinh lôgic của game:

Game này chơi chủ yếu KT va chạm, chúng ta cần 1 biến kiểu bool để làm điều đó, tất nhiên khi begin, chưa có va chạm nên biến này nhận false.

KT va chạm thông qua 1 hcn bao quanh object Ship, chúng ta đã viết hàm GetBound đúng không nào, từ đó tạo ra hcn bao quanh ship là việc rất dễ dàng.

Foreach thì chắc các bạn bít rùi, nó là vòng for và lặp trong 1 mảng cụ thể. Foreach lặp trong mảng Components và tìm những object là Meteor, KT va chạm với Ship, nếu có thì bùm, xoá sạch meteor và bắt đầu lại.

Trong thời gian chạy game, sau 1 khoảng thời gian `ADDMETEORTIME` Hàm `CheckforNewMeteor();`

Sẽ tạo ra một meteor.

```
private void DoGameLogic()
        {
            // Check collisions
            bool hasColision = false;
            Rectangle shipRectangle = player.GetBounds();
            foreach (GameComponent gc in Components)
            {
                if (gc is Meteor)
                {
                    hasColision = ((Meteor)gc).CheckCollision(shipRectangle);
                    if (hasColision)
                    {
                        // BOOM!
                        explosion.Play();
                        // Remove all previous meteors
                        RemoveAllMeteors();
                        // Let's start again
                        Start();

                        break;
                    }
                }
            }

            // Add a new meteor if is time
            CheckforNewMeteor();
        }
```

Hàm Update :Cực kỳ đơn giản, tạo ra player nếu chưa có và thực hiện hàm `DoGameLogic();` Nó đã chứa toàn bộ game play của chúng ta.

```
protected override void Update(GameTime gameTime)
        {
            keyboard = Keyboard.GetState();

            // Start if not started yet
            if (player == null)
            {
                Start();
            }

            // Fun never ends..
            DoGameLogic();

            // Update all other components
```

```
            base.Update(gameTime);
        }
```

Hàm đồ hoạ Draw:

Chủ yếu chúng ta tạo ra 2 texture, 1 cái làm hình nền `backgroundTexture`.

Ship và Meteor ko cần code bởi lẽ chúng ta đã code trong class của chúng, chúng tự đc vẽ khi run Game.

In ra số lượng Meteor nhằm mục đích cho người chơi biết "level" của họ, cũng như time bám trụ đc, tất nhiên KQ cuối cùng player sẽ thua thôi vấn đề là họ trụ đc bao lâu.String là chuỗi, lệnh `rockCount.ToString()` đưa SL meteor thành kiểu chuỗi và in ra theo 1 hàm in chuỗi ký tự

```
spriteBatch.DrawString(gameFont, "Rocks: " + rockCount.ToString(), new
Vector2(15, 15), Color.YellowGreen);
            spriteBatch.End();
```

```
protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.Black);

            // Draw background texture in a seperate pass
            spriteBatch.Begin();
            spriteBatch.Draw(backgroundTexture, new Rectangle(0, 0,
                graphics.GraphicsDevice.DisplayMode.Width,
                graphics.GraphicsDevice.DisplayMode.Height),
                Color.LightGray);
            spriteBatch.End();
            // Start rendering sprites
            spriteBatch.Begin(SpriteBlendMode.AlphaBlend);
            // Draw the game components (sprites included)
            base.Draw(gameTime);
            // End rendering sprites
            spriteBatch.End();
            // Draw Score
            spriteBatch.Begin();
            spriteBatch.DrawString(gameFont, "Rocks: " +
rockCount.ToString(), new Vector2(15, 15), Color.YellowGreen);
            spriteBatch.End();
        }
    }
```

Xong F5 đi `J`

Lession sau sẽ sử dụng lại project này, các methods sẽ sử dụng lại nhiều đấy. Dự án nâng cấp sẽ thuộc loại cực kỳ phức tạp. các class không chỉ viết cho Object mà còn cả việc quản lý âm thanh, object và làm nền cho các class khác + nhiều phương thức mới sẽ đc introduced..., Bạn nên chuẩn bị tư tưởng + chút kiên nhẫn, ko phải chém gío đâu`L`

Z z z...

BÀI 6 TÍNH KẾ THỪA - NÂNG CẤP DỰ ÁN.

Phần I: Tính kế thừa (Inheritance):Phủ nhận-Tạo mới.

Nếu đã rành C# xin mời qua luôn phần 2 J ! chưa rành lắm thì học lại C# rồi đọc phần dưới L :

Tính kế thừa là gì? Chả phải bạn đã sử dụng nó ngay từ khi code dòng đầu tiên trong XNA sao? Giả xử đối tượng A kế thừa từ đối tượng B chúng ta sẽ khai báo:

Class A:B

{

//Nội dung class

}

Rất đơn giản đúng không, cũng như chúng ta coded rằng:

```
public class Game1 : Microsoft.Xna.Framework.Game
```

Có nghĩa là game của chúng ta đã kế thừa từ lớp `Microsoft.Xna.Framework.Game` Sẵn có của XNA, Game1 thừa hưởng mọi thuộc tính, hàm, phương thức... của mẹ nó là `Microsoft.Xna.Framework.Game.`

Tính kế thừa rất tiện lợi cho những dự án lớn, nó làm giảm khối lượng công việc của programmer giúp chương trình mạch lạc, đơn giản hơn, dễ debug và kiểm soát. Nói chay khó hiểu mình nếu thử VD này. Bạn có 3 chủng loại monster khác nhau trong game của bạn, chúng đều biết chạy đến chỗ bạn, tấn công và bỏ chạy khi sắp die, thế nếu chúng ta tạo 1 lớp chung cho 3 chủng monster trên gọi là **Monster**, sau đó tạo riêng cho từng con mỗi con một class riêng kế thừa từ lớp **Monster.** Chúng ta chỉ cần code cho con 1 là melee, con 2 là range, con 3 là boss có skill chưởng chả hạn, khi có lỗi, cũng dễ khoanh vùng mà debug và chương trình cũng gọn hơn rất nhiều.

Với VD nêu trên, lớp **Monster** sẽ đc khai báo như sau:

```
public abstract class Monster
```

{

}

Cái chữ abstract chắc các bạn đã biết, nó đánh dấu lớp ảo, tức là lớp làm nền cho các lớp khác kế thừa, lớp ảo không tạo ra đối tượng khi chạy chương trình, nếu cố tình làm điều đó, C# sẽ báo lỗi.

Khi mình cần code class cho boss, code sẽ như sau:

```
Public class Boss:Monster
```

{

}

Tượng tự Boss sẽ có tất cả các biến, hàm, phương thức của Monster, như trong một số trường hợp, đó ko fải là điều bạn mong muốn. VD: con lính sẽ chạy khi sắp die nhưng con boss sẽ tử chiến đến cùng, khi đó bạn sẽ phải viết lại hàm bỏ chạy của monster:

Trong class của monster:

```
public virtual void bỏ chạy()
```

Trong class của Boss:

```
public override void bỏ chạy()
```

chữ virtual đánh dấu phương thức ảo, tức là trong tuỳ trường hợp cụ thể, nó có thể bị đè xuống (tức là đi họp luôn) bởi phương thức đc đánh dấu `override` , phương thức mới phủ nhận hoàn toàn phương thức cũ

nếu bạn cần lại cái gì từ phương thức cũ, lệnh base sẽ làm điều đó. Ko dùng base cho phương thức đc đánh dấu override

VD, trong lớp Monster:

Public void bochay(int Speed)

Bạn muốn giữ lại speed cho boss, trong lớp Boss:

Public void bochay(int Speed):base(`Speed)`

Bạn có thể tạo ra một phương thức mới bằng từ khoá new, đều này không phủ nhận gì từ lớp mẹ (lớp cơ sở, lớp cho kế thừa), ngược lại với abstract là sealed... nhiều lắm, các bạn tự tìm hiểu thêm cái này vì nó là C# thuần tuý.

Một phần nữa mà nhẽ ra mình ko giải thích, nhưng project dùng nhiều quá, đó là bộ truy cập dữ liệu (ví dụ của class Z):

```
public “KIỂU DỮ LIỆU” “TÊN BIẾN X”
        {
            get { return “Y LÀ GIÁ TRỊ BIẾN X SẼ NHẬN”; }
            set { “BIÊN Y” = “THIẾT LẬP GIÁ TRỊ Y SAU KHI X NHẬN GIÁ TRỊ”; }
        }
```

Nhận xét:

X với Y nên có sự tương đồng cho dễ nhớ và chương trình tường minh, VD: frames với Frames. Nó đảm bảo sự hài hoà giữa lớp cơ sở và lớp mở rộng, hay giữa các lớp khác nhau Phần get bắt buộc có, để BIẾN có giá trị trả về, phần set dùng để thay đổi biến X đó nếu không có set, biến chỉ có thể đọc giá trị (read only), thường chúng ta để set { “BIÊN Y” = value; } Đê thông tin trao đổi 2 chiều

Chức năng: Cho phép một số biến đánh dấu là protected (chỉ cho phép truy cập từ object của lớp đó hoặc lớp con của nó) hay private (chỉ có object của lớp đó mới truy cập đc) có thể truy cập bởi các lớp khác (tên biến đánh dấu public ..).

Cách sử dụng: như đã nói, cái đc dùng phổ biến với dạng là: khi chúng ta cần truy cập biến (thuộc tính) X của lớp Z mà lại đang ở lớp T chả hạn. Bây giờ lớp T có thuộc tính M, mà X muốn lấy thống tin từ M từ lớp T, ta viết mã lệnh ở T đơn giản như sau:

```
Z.X = M;
```

Tất nhiên object lớp Z đã đc khai báo ở lớp T như một object (VD: khi Ship ở màn action đc khai báo tức là Ship.Speed đã dc khai báo (với Speed đc cái đặt bộ truy cập dữ liệu)

Cách nhận dang: Bộ truy cập này đc dùng phổ biến có chỉ hết trong bài cũng khó, các bạn có thể dựa vào cách đặt tên biến: Các programmer khi LT thường đặt tên biến của lớp không viết hoa ở đầu (VD: speed), còn biến để truy cập từ lớp ngoài thì ngược lại (Speed). Cũng như là các LTV thường hiểu ngầm Methods, thuộc tính sẵn có thì dùng chữ viết hoa cho các chư của tên của nó, (VD: SpeedLower) còn cái do LTV viết, thì họ viết kiểu thường cho chữ đầu, sau đó chữ hoa và lại chữ thường... (VD: speedLower)

PHẦN II: NÂNG CẤP DỰ ÁN

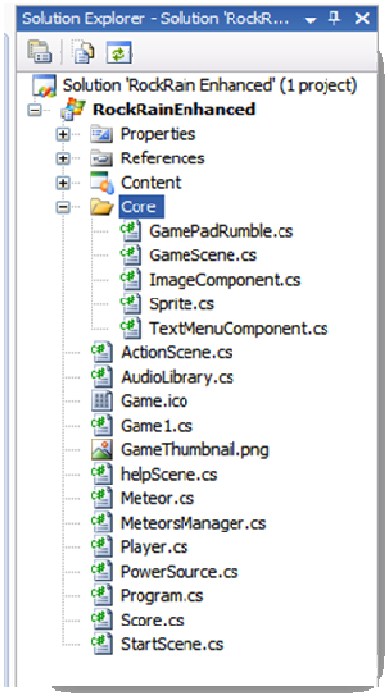
Dự án của chúng ta dựa vào tính kế thừa, do đó, sẽ có 2 loại class (trừ thằng Game1.cs ra)

-Core:các lớp cho phép các object khác trong game kế thừa.(sprite, màn ...), chúng đc gom trong một folder

-Class còn lại, class của các object đc thể hiện(ship;meteor,màn start, màn play, màn end)...

Dự án nâng cấp sẽ có các phương pháp trong việc tạo ra hoạt hình sau 1 khoảng thời gian (sẽ giải thích sau). Chúng ta học cách làm hoạt hình cho sprite thông qua việc tạo ra và cho chạy các frame của nó. 1 Game Hoàn chỉnh cần có màn begin, play và end, đây cũng là khởi đầu của mọi rắc rối J

ĐÂY là SL công việc fải làm L :



OK! bắt đầu nào...

A. Class Nhân (core)

1.Sprite.cs

Bắt đầu từ Sprite.cs, đây là nền cho các object như PowerSource (cái giúp game kéo dài hơn) và Meteor hoạt động.

Using và kế thừa ko có j giải thích, chỉ lưu ý namespace khi đặt sprite.cs trong folder core thôi:

```
#region Using Statements

using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

#endregion

namespace RockRainEnhanced.Core
{
    /// <summary>
    /// </summary>
```

```
public class Sprite : DrawableGameComponent
```

Khai báo:

```
        private int activeFrame;
```

`activeFrame` Biến xác định index(chỉ số) của frame đã chạy. Frame là khung hình chứa 1 trạng thái của sprite nào đó, chúng thay đổi rất nhanh trên màn hình và làm chúng ta có cảm giác là vật đang chuyển động (hoạt hình). XNA chạy hoạt hình cho sprite cũng dựa vào các khung hình frame của sprite đó, làm cho chúng ta cảm giác sprite đc chạy.cũng Tương tự cho việc làm phim hoạt hình hay khi bạn sử dụng flash để design.

```
        private readonly Texture2D texture;
        private List<Rectangle> frames;
```

`frames` Là biến danh sách các HCN chính là các cạnh của khung hình frame (là các trạng thái của sprite), nó cũng ngang ngang như Mảng chứa các frame vậy.

```
        protected Vector2 position;
        protected TimeSpan elapsedTime = TimeSpan.Zero;
```

`elapsedTime` Là biến xác định thời điểm trong game, đc thiết lập là 0 (zero)

```
        protected Rectangle currentFrame;
```

`currentFrame` Là HCN bao quanh frame hiện thời mà XNA đang duyệt

```
        protected long frameDelay;
```

`frameDelay` Là biến chỉ thời gian chuyển từ frame này sang frame kế nó

```
        protected SpriteBatch sbBatch;
```

Các bạn có thể quan sát texture của Meteor để đếm nó có bao nhiêu frame :D

Có 3 loại Metẻor và mỗi loại có 8 cái frame.

Hàm nhập dữ liệu:

```
public Sprite(Game game, ref Texture2D theTexture)
            : base(game)
        {
            texture = theTexture;
            activeFrame = 0;
        }
```

Giống như project cũ, sprite nhập vào nó texture và game, thiết lập cho Frameactive = 0; (frame đầu tiên)

```
public List<Rectangle> Frames
        {
            get { return frames; }
            set { frames = value; }
        }
```

Code trên gọi là 1 bộ truy cập dữ liệu cho biến Frames kiểu `List<Rectangle>` trong C#, khiến cho nó luôn phải nhận về giá trị của `frames` .

Hàm thiết lập:

```
 public override void Initialize()
        {
            // Get the current spritebatch
            sbBatch = (SpriteBatch) Game.Services.GetService(typeof
(SpriteBatch));
```

```
            base.Initialize();
        }
```

sbBatch đc chuyển xuống hàm này, thực ra để nó ở phần nhập dữ liệu cũng chả sao, đều đc gọi khi chạy game cả.

Hàm Update():

```
public override void Update(GameTime gameTime)
        {
            elapsedTime += gameTime.ElapsedGameTime;

            // it's time to a next frame?
            if (elapsedTime > TimeSpan.FromMilliseconds(frameDelay))
            {
                elapsedTime -= TimeSpan.FromMilliseconds(frameDelay);
                activeFrame++;
                if (activeFrame == frames.Count)
                {
                    activeFrame = 0;
                }
                // Get the current frame
                currentFrame = frames[activeFrame];
            }

            base.Update(gameTime);
        }
```

Hàm này thực hiện các công việc sau: Tính thời gian chay game thông qua

```
elapsedTime += gameTime.ElapsedGameTime;
```

Nếu thời gian nhiều hơn thời gian trong framêdelay thì câu lệnh

```
elapsedTime -= TimeSpan.FromMilliseconds(frameDelay);

activeFrame++;
                if (activeFrame == frames.Count)
                {
                    activeFrame = 0;
                }
```

Biến elapsedTime sẽ nhận giá trị vô cùng bé (-> 0) do hàm update chạy và duyệt rất nhanh, khi biến này mới tăng cao hơn framedelay 1 tẹo nó se thực hiện câu lệnh

```
elapsedTime -= TimeSpan.FromMilliseconds(frameDelay);
```

Và elapsedTime chỉ còn lại giá trị tăng cao hơn 1 tẹo đó, lệnh `TimeSpan.FromMilliseconds(frameDelay)` trả về giá trị của framedelay theo miligiây.

Nó tăng số framêActive lên tức là để frame có thể chạy nhằm thực hiện hoạt hình cho sprite, đoạn sau ráp texture vào bạn sẽ thấy rõ hơn.`frames.Count` Trả về sl frame của sprite, nếu

`activeFrame == frames.Count` tức là đã chạy hết frame của sprite và hoạt hình sẽ đc bắt đầu lại.
Hàm đồ hoạ

```
public override void Draw(GameTime gameTime)
        {
            sbBatch.Draw(texture, position, currentFrame, Color.White);

            base.Draw(gameTime);
        }
```

Câu lệnh currentFrame = frames[activeFrame]; Draw luôn chỉ vẽ frame hiện thời. khi qua frame khác currentFrame thay đổi và KQ là `sbBatch.Draw(texture, position, currentFrame, Color.White)` cũng thay đổi từ đó sprite đã đc hoạt hình.

2.Thành phần Image:

Cái class này đơn giản, nó chỉ thuần tuý thực hiện các công việc sau:

-Là Đ/n cho các thành phần image: ảnh nền của các màn

```
public enum DrawMode
        {
            Center = 1,
            Stretch,
        } ;
```

Đây là dãy các kiểu vẽ image, nó đc dùng khi vẽ bảckground

Phần khai báo

drawMode có kiểu DrawMode nên nó có 2 thành phần drawMode.`Center và drawMode.Center; và` drawMode.`Center` có giá trị 1.

```
        // Texture to draw
        protected readonly Texture2D texture;
        // Draw Mode
        protected readonly DrawMode drawMode;
        // Spritebatch
        protected SpriteBatch spriteBatch = null;
        // Image Rectangle
        protected Rectangle imageRect;
```

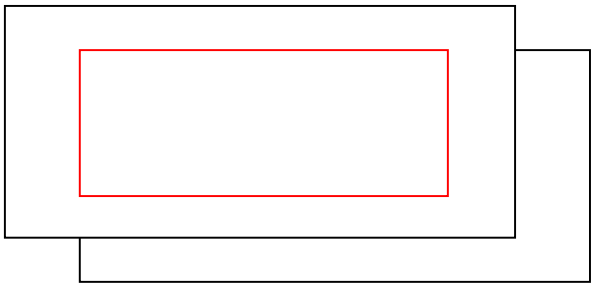
imageRect Là HCN bao quanh imageComponent. Kỹ thuật dùng HCN khoanh vùng lấy texture trên một bức ảnh ta import vào đã quen thuộc ở lession trước, bạn cũng thấy nó ở hàm Draw(GameTime gameTime)

Hàm nhập data: gồm game, Texture cho imageComponent và kiểu vẽ nó.

Ứng với các kiểu vẽ khác nhau là các HCN bao quanh khác nhau, từ đó đưa ra các vùng khác nhau trên texture và có các texture khác nhau.

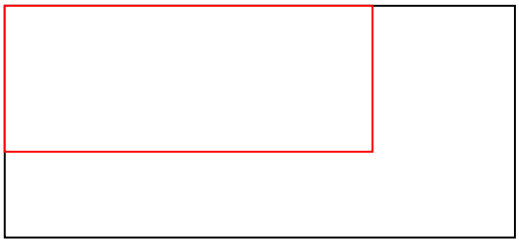
Center: Hình CN nằm chính giữa Texture Background. Nếu để ý các bạn sẽ thấy texture của chúng ta khác nhau và khác size of screen, chính xác là các BackGrround đều lớn hơn screen.

Việc dùng Rectangle lấy texture ở chính giữa các texture background giúp chúng ta có hình ảnh "thật" của texture hơn thôi, chứ ko thể đem hết cái texture background to thù lù đó vô screen đc,



HCN to tương trưng cho image dùng làm texture, HCN to nằm phía dưới, lệch bên phải tượng trương vùng lấy texture, do do Screen < texture nên chỉ có HCN nhỏ tượng trưng cho HCN bao quanh phần sẽ lấy làm background khi dùng kiểu center và HCN có kich thước = screen

Stretch: HCN có toạ độ 0;0 trên texture, và có kick thước = screen, khi đó HCN sẽ ở vị trí như sau, sau đó nó lấy vùng texture đó làm background:



Vì sao phức tạp như vậy, cái này cũng tuỳ trường hợp thôi, vì texture kick thước to quá, có cái lấy ở tâm image mới có texture đẹp, có cái lấy ở cạnh thôi cũng đẹp...

```
public ImageComponent(Game game, Texture2D texture, DrawMode drawMode)
            : base(game)
        {
            this.texture = texture;
            this.drawMode = drawMode;
            // Get the current spritebatch
            // this chính là ImageComponent
            spriteBatch = (SpriteBatch)
                Game.Services.GetService(typeof (SpriteBatch));

            // Create a rectangle with the size and position of the image
            switch (drawMode)
            {
                case DrawMode.Center:
                    imageRect = new Rectangle(
(Game.Window.ClientBounds.Width-texture.Width)/2,
(Game.Window.ClientBounds.Height - texture.Height)/2,
texture.Width,
texture.Height);
```

```
                    break;
                case DrawMode.Stretch:
                    imageRect = new Rectangle(0, 0,
Game.Window.ClientBounds.Width,
Game.Window.ClientBounds.Height);
                    break;
            }
        }

        /// <summary>
        /// Allows the game component to draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>
        public override void Draw(GameTime gameTime)
        {
            spriteBatch.Draw(texture, imageRect, Color.White);
            base.Draw(gameTime);
        }
    }
}
```

3.Class cho thành phần Menu

Một Menu đc tạo ra hoàn toàn = code, tức là ko có tẹo paint, photoshop hay những cái đại loại thế đâu nhá.

Chúng ta tạo ra Menu để gamer lựa chọn: 1 player, 2 players, help, out => chúng có 4 cái => Menu của chúng ta có 4 thành phần. User dùng phím Mũi tên để chọn 1 trong 4.

Chào buổi sáng của 1 lớp như thường lệ:

```
#region Using Statements

using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

#endregion

namespace RockRainEnhanced.Core
{
    /// <summary>
    /// This is a game component that implements a menu with text elements.
    /// </summary>
    public class TextMenuComponent : DrawableGameComponent
    {
```

Phần khai báo.

ở thằng SpriteBatch các bạn thấy chữ null ko, null tức là trống rỗng mà vốn dĩ SB cũng là 1 sprite rỗng rồi, câu lệnh này chỉ làm NNgữ thêm tường minh, thế thôi.

```
        // Spritebatch
```

```
        protected SpriteBatch spriteBatch = null;
 Sử dụng font chữ, cái regular dùng cho list trong Menu, cái selected dùng
cho phần đc chọn trong menu (ta chọn = phím mũi tên)
        protected readonly SpriteFont regularFont, selectedFont;
        // Colors
Quy định màu cho 2 loại font
        protected Color regularColor = Color.White, selectedColor =
Color.Red;
        // Menu Position

Vị trí Menu
        protected Vector2 position = new Vector2();
        // Items
Biến ghi lại phần nào trong menu đc chọn, mặc định là 0 (là cái đầu tiên đó)
        protected int selectedIndex = 0;
Biến ghi lại tên các thành phần của menu
        private readonly List<string> menuItems;
Khởi tạo bàn phím
        // Used for handle input
        protected KeyboardState oldKeyboardState;
Khai báo size
        // Size of menu in pixels
        protected int width, height;
Thêm hiệu ứng âm thanh khi chọt vô
        // For audio effects
        protected AudioLibrary audio;
 -> Chúng ta có 1 class riêng về AudioLibrary, Sẽ trình bày sau
```

Phần nhập data:

Menu cần nhập về game, 2loại font

Từ khoá `this` chỉ là chính đối tượng của class Menu này cần nhập `selectedFont`. Từ khoá thí dùng để phân biệt selectedFont của class và của tham số tham chiếu selectedFont trong hàm nhập data

```
public TextMenuComponent(Game game, SpriteFont normalFont,
            SpriteFont selectedFont)
            : base(game)
        {
            regularFont = normalFont;
            this.selectedFont = selectedFont;
Khởi tạo giá trị (vẫn là null):
            menuItems = new List<string>();

            // Get the current spritebatch
            spriteBatch = (SpriteBatch)
                Game.Services.GetService(typeof (SpriteBatch));

            // Get the audio library
            audio = (AudioLibrary)
                Game.Services.GetService(typeof(AudioLibrary));

            // Used for input handling
            oldKeyboardState = Keyboard.GetState();        }
```

```
        /// <summary>
        /// Set the Menu Options
        /// </summary>
        /// <param name="items"></param>
```

Hàm dưới đây làm nhiệm vụ làm sạch Menu, đặt Nội dung cho Menu, sau đó tính size của từng Thành hần trong Menu, tức là cứ mỗi thành phần khi hàm đc gọi sẽ có width và height.

```
        protected void CalculateBounds()
        {
            width = 0;
            height = 0;
            foreach (string item in menuItems)
//duyệt từng chuỗi(gọi là nhãn cũng đc label) trong Menu
            {
                Vector2 size = selectedFont.MeasureString(item);
 //Chiều dài chuỗi ký tự nhập vào
                if (size.X > width)
//Tính width theo size.X (chiều dài của label trong menu)
                {
                    width = (int) size.X;
                }
                height += selectedFont.LineSpacing;
//Chiều rộng height tính theo khoảng cách hàng
            }
        }
```

string[] items: items là mảng các string, và chúng ta cần 4 chuỗi.

```
        public void SetMenuItems(string[] items)
        {
            menuItems.Clear();
            menuItems.AddRange(items);
            CalculateBounds();
        }

        /// <summary>
        /// Width of menu in pixels
        /// </summary>
```

Các Bộ truy cập dữ liệu, luôn trả về giá trị của width cho Width và `Height`

Lệnh set chỉ để quy định giá trị đó là `value của chính nó` nó ko quan trọng đâu.

```
public int Width
        {
            get { return width; }
        }

        /// <summary>
        /// Height of menu in pixels
        /// </summary>
        public int Height
```

```
        {
            get { return height; }
        }

        /// <summary>
        /// Selected menu item index
        /// </summary>
        public int SelectedIndex
        {
            get { return selectedIndex; }
            set { selectedIndex = value; }
        }

        /// <summary>
        /// Regular item color
        /// </summary>
        public Color RegularColor
        {
            get { return regularColor; }
            set { regularColor = value; }
        }

        /// <summary>
        /// Selected item color
        /// </summary>
        public Color SelectedColor
        {
            get { return selectedColor; }
            set { selectedColor = value; }
        }

        /// <summary>
        /// Position of component in screen
        /// </summary>
        public Vector2 Position
        {
            get { return position; }
            set { position = value; }
        }
```

Hàm Update(): nơi mọi thứ Hoạt động:

Bạn thấy chữ release có quen ko? Thế chữ Press thế nào? Còn Drop?

Release hiểu đơn giản là 1 sự kiện (event) diễn ra khi bạn đã Press 1 phím và ngay sau đó drop nó ra, chúng ta tìm hiểu kỹ thuật release phím trong XNA, VD như:

```
down = (oldKeyboardState.IsKeyDown(Keys.Down) &&
                (keyboardState.IsKeyUp(Keys.Down)));
```

Down trả về true khi có event release bằng một phép toán && (tức là ***and*** đó)

```
        public override void Update(GameTime gameTime)
        {
            //Khởi tạo 2 bàn phím
```

```
            KeyboardState keyboardState = Keyboard.GetState();
            oldKeyboardState = keyboardState;

            bool down, up;
            // Handle the keyboard
            down = (oldKeyboardState.IsKeyDown(Keys.Down) &&
                (keyboardState.IsKeyUp(Keys.Down)));
            up = (oldKeyboardState.IsKeyDown(Keys.Up) &&
                (keyboardState.IsKeyUp(Keys.Up)));

            if (down || up)
            {
                audio.MenuScroll.Play();
            }
// nếu release phím down (mũi tên hướng xuống)
            if (down)
            {
//tăng chỉ số của thành phần menu đc chọn lên 1 (chọn Nội dung khác trong
//Menu), nếu chọn hết thì trở về Cái list đầu tiên (có index = 0)
                selectedIndex++;
                if (selectedIndex == menuItems.Count)
                {
                    selectedIndex = 0;
                }
            }
//Ngược với cái trên
            if (up)
            {
                selectedIndex--;
                if (selectedIndex == -1)
                {
                    selectedIndex = menuItems.Count - 1;
                }
            }



            base.Update(gameTime);
        }
```

Hàm Đồ hoạ:

Hàm hoạt động như sau: gán position.Y (toạ độ y của Menu) cho y nhằm mục đích mỗi component của menu đều có y (y là biến địa phương trong hàm Draw) toạ độ x thì không cần vì mỗi component đều chung nhau toạ độ x, (thẳng hàng với nhau theo chiều dọc)

Duyệt từng thành phần của Menu

Kt i đang có index mấy để thiết đặt cho nó font và color.

```
        public override void Draw(GameTime gameTime)
        {
            float y = position.Y;
            for (int i = 0; i < menuItems.Count; i++)
            {
```

```
                SpriteFont font;
                Color theColor;
                if (i == SelectedIndex)
                {
                    font = selectedFont;
                    theColor = selectedColor;
                }
                else
                {
                    font = regularFont;
                    theColor = regularColor;
                }
```

Draw text có 2 loại, cái bóng có chữ màu đen đc vẽ trước, bởi lẽ cái j vẽ trước sẽ bị đè xuống bởi cái sau, chính là cái text chính thức của menu, không phải bóng của nó.

```
                // Draw the text shadow
                spriteBatch.DrawString(font, menuItems[i],
                    new Vector2(position.X + 1, y + 1), Color.Black);
                // Draw the text item
                spriteBatch.DrawString(font, menuItems[i],
                    new Vector2(position.X, y), theColor);
                y += font.LineSpacing;
            }

            base.Draw(gameTime);
        }
    }
}
```

4.Lớp Game Scene – Từng Màn trong game:

Đây là nền cho các class sau: màn begin, play, help

```
#region Using Statements

using System.Collections.Generic;
using Microsoft.Xna.Framework;

#endregion

namespace RockRainEnhanced.Core
{
    /// <summary>
    /// This is the base class for all game scenes.
    /// </summary>
```

Lớp đc đánh dấu abstract đây là lớp ảo (còn gọi là trừu tượng)

```
    public abstract class GameScene : DrawableGameComponent
    {
        /// <summary>
        /// List of child GameComponents
        /// </summary>
```

Trong Game1, do tính kế thừa mà nó có Mảng chứa component nhưng GameScene thì ko, nó phải tự tạo ra danh sách riêng chứa các component của nó.

```
        private readonly List<GameComponent> components;

base miễn giải thích
        public GameScene(Game game)
            : base(game)
        {
//dach sách components hiện tại vẫn là null
            components = new List<GameComponent>();
//Mặc định Scene này ko hiển thị và ko hoạt động, đây là 2 thuộc tính kế thừa
tư DrawableGameComponent nên khởi tạo
            Visible = false;
            Enabled = false;
        }
```

Hàm dưới cho phép Scene hiển thị và hoạt động

```
        public virtual void Show()
        {
            Visible = true;
            Enabled = true;
        }

        /// <summary>
        /// Hide the scene
        /// </summary>
```

Hàm dưới này ngược với hàm trên

```
public virtual void Hide()
        {
            Visible = false;
            Enabled = false;
        }

        /// <summary>
        /// Components of Game Scene
        /// </summary>
```

Bộ truy cập data

```
public List<GameComponent> Components
        {
            get { return components; }
        }
```

Hàm Update(): Update từng thành phần rất chi là "trừu tượng" chúng ta ko hề biết `components.Count` Là bao nhiêu? Cái này tuỳ thuộc từng scene cụ thể. Update cho những thành phần đc phép hoạt động (enable = true).

```
    public override void Update(GameTime gameTime)

        {
            // Update the child GameComponents
            for (int i = 0; i < components.Count; i++)
            {
                if (components[i].Enabled)
```

```
                {
                    components[i].Update(gameTime);
                }
            }

            base.Update(gameTime);
        }
```

Hàm đồ hoạ: Vẽ từng component của scene nếu nó đc phép hiển thị (visible = true) và nếu nó thuộc lớp `DrawableGameComponent` hay kế thừa từ lớp này. Tất nhiên cái này chỉ là gọi hàm, còn Draw ntn (texture, chế độ vẽ...) là do class của từng component quyết định

```
        public override void Draw(GameTime gameTime)
        {
            // Draw the child GameComponents (if drawable)
            for (int i = 0; i < components.Count; i++)
            {
                GameComponent gc = components[i];
                if ((gc is DrawableGameComponent) &&
                    ((DrawableGameComponent) gc).Visible)
                {
                    ((DrawableGameComponent) gc).Draw(gameTime);
                }
            }
            base.Draw(gameTime);
        }
    }
}
```

Cơ bản là xong phần class nhân (core), sau đây sẽ là các class cho các object thể hiện trong game

B. Class Thể hiện (Extended)

1.AudioLibrary

Đi từ cái đơn giản nhất: Âm thanh:

Tất cả sound của game đc bỏ vào đây, đầu tiên là khai báo (song-nhạc nền và Effect-Hiệu ứng)

Sau đó chúng đc tạo bộ truy cập dữ liệu Cho các màn trong game, các sự kiện diễn ra có thể sử dụng sound khi chạy game (khởi động, ăn điểm, va chạm...) và 1 hàm load sound khi khởi động.

```
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Media;

namespace RockRainEnhanced
{
    public class AudioLibrary
    {
```

```
private SoundEffect explosion;
private SoundEffect newMeteor;
private SoundEffect menuBack;
private SoundEffect menuSelect;
private SoundEffect menuScroll;
private SoundEffect powerGet;
private SoundEffect powerShow;
private Song backMusic;
private Song startMusic;

public SoundEffect Explosion
{
    get { return explosion; }
}

public SoundEffect NewMeteor
{
    get { return newMeteor; }
}

public SoundEffect MenuBack
{
    get { return menuBack; }
}

public SoundEffect MenuSelect
{
    get { return menuSelect; }
}

public SoundEffect MenuScroll
{
    get { return menuScroll; }
}

public SoundEffect PowerGet
{
    get { return powerGet; }
}

public SoundEffect PowerShow
{
    get { return powerShow; }
}

public Song BackMusic
{
    get { return backMusic; }
}

public Song StartMusic
{
    get { return startMusic; }
}

public void LoadContent(ContentManager Content)
{
```

```
            explosion = Content.Load<SoundEffect>("explosion");
            newMeteor = Content.Load<SoundEffect>("newmeteor");
            backMusic = Content.Load<Song>("backMusic");
            startMusic = Content.Load<Song>("startMusic");
            menuBack = Content.Load<SoundEffect>("menu_back");
            menuSelect = Content.Load<SoundEffect>("menu_select3");
            menuScroll = Content.Load<SoundEffect>("menu_scroll");
            powerShow = Content.Load<SoundEffect>("powershow");
            powerGet = Content.Load<SoundEffect>("powerget");
        }
    }
}
```

2.Lớp Meteor – ROCKS OF SPACE:

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RockRainEnhanced.Core;

namespace RockRainEnhanced
{
    /// <summary>
    /// This class is the Animated Sprite for a Meteor
    /// </summary>
    public class Meteor : Sprite
    {
```

Meteor kế thừa từ sprite. Nó có tất cả ....

Các thuộc tính mới cho Meteor:Tốc độ, khả năng tạo ra số ngẫu nhiên và index cho mã số cho từng Meteor vì chúng SL rất nhiều.

```
        // Vertical velocity
        protected int Yspeed;
        // Horizontal velocity
        protected int Xspeed;
        protected Random random;
        // Unique id for this meteor
        private int index;
```

Hàm nhập Data:

Frames lấy dữ liệu từ frames thông qua bộ truy cập, frame là HCN bao quanh lấy Texture của Meteor trên cái image dùng làm texture chung to đùng đoàng.

```
        public Meteor(Game game, ref Texture2D theTexture) :
            base(game, ref theTexture)
        {
            Frames = new List<Rectangle>();
//Hiện tại Frames vẫn null
//Câu lệnh dưới Add frame vào Frames có kiểu List
//Toạ độ Y thay đổi tức là chúng ta lấy frame tuần tự từ trên -> dưới
            Rectangle frame = new Rectangle();
            frame.X = 468;
            frame.Y = 0;
```

```
            frame.Width = 49;
            frame.Height = 44;
            Frames.Add(frame);

            frame.Y = 50;
            Frames.Add(frame);

            frame.Y = 98;
            frame.Height = 45;
            Frames.Add(frame);

            frame.Y = 146;
            frame.Height = 49;
            Frames.Add(frame);

            frame.Y = 200;
            frame.Height = 44;
            Frames.Add(frame);

            frame.Y = 250;
            Frames.Add(frame);

            frame.Y = 299;
            Frames.Add(frame);

            frame.Y = 350;
            frame.Height = 49;
            Frames.Add(frame);

            // Initialize the random number generator and put the meteor in
your
            // start position
            random = new Random(GetHashCode());
            PutinStartPosition();
        }
```

Hàm đặt vận tốc và toạ độ ngẫu nhiên một khi Meteor đc tạo ra:

```
        public void PutinStartPosition()
        {
            position.X = random.Next(Game.Window.ClientBounds.Width -
                currentFrame.Width);
            position.Y = 0;
            YSpeed = 1 + random.Next(9);
            XSpeed = random.Next(3) - 1;
        }
```

Bộ truy cập dữ liệu

```
        public int YSpeed
        {
            get { return Yspeed; }
            set
            {
                Yspeed = value;
                frameDelay = 200 - (Yspeed * 5);
//Đặt thời gian chuyển frame kế nó nhằm tạo hoạt hình
```

```
            }
        }

        /// <summary>
        /// Horizontal Velocity
        /// </summary>
        public int XSpeed
        {
            get { return Xspeed; }
            set { Xspeed = value; }
        }

        /// <summary>
        /// Meteor Identifier
        /// </summary>
        public int Index
        {
            get { return index; }
            set { index = value; }
        }
```

Hàm Update:

Nếu meteor ra khỏi màn hình, đặt nó vào vị trí cũ., nếu không thì move nó đi.

```
        public override void Update(GameTime gameTime)
        {
            // Check if the meteor still visible
            if ((position.Y >= Game.Window.ClientBounds.Height) ||
                (position.X >= Game.Window.ClientBounds.Width) ||
                (position.X <= 0))
            {
                PutinStartPosition();
            }

            // Move meteor
            position.Y += Yspeed;
            position.X += Xspeed;

            base.Update(gameTime);
        }
```

Kiểm tra va chạm với 1 HCN (tất nhiên HCN bao quanh Ship rồi) với HCN bao quanh Mêteor nhằm biết khi nào bùm hoặc restart game.

```
        public bool CheckCollision(Rectangle rect)
        {
            Rectangle spriterect =new Rectangle((int) position.X, (int)
position.Y,
                currentFrame.Width, currentFrame.Height);
            return spriterect.Intersects(rect);
        }
    }
}
```

3. Class quản Lý các Meteor: Tạo ra, KT va chạm, gọi đồ hoạ...

Class có vị trí rất quan trong, tự nó tạo ra, điều khiển các Meteor trong chương trình, đoạn sau ráp vô thì biến meteors khai báo kiểu `MeteorsManager` rất gọn

```
#region Using Statements

using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

#endregion

namespace RockRainEnhanced
{
    /// <summary>
    /// This game component implements a manager for all Meteors in the game.
    /// </summary>
    public class MeteorsManager : DrawableGameComponent
    {
```

Phần Khai báo biến

Danh sách các Meteor đg run game.

```
        protected List<Meteor> meteors;
```

Hằng số SL Các meteor khi begin

```
    private const int STARTMETEORCOUNT = 10;
```

Khoảng Time để tạo thêm Meteor đơn vị miligiây (là 5s)

```
        private const int ADDMETEORTIME = 5000;
```

Âm thanh

```
        private AudioLibrary audio;
```

Texture

```
        protected Texture2D meteorTexture;
```

Thời gian chạy game , begin với 0. Dùng để tạo ra Meteor sau 1 khoảng 5 s

```
        protected TimeSpan elapsedTime = TimeSpan.Zero;
```

Hàm nhập data:

```
        public MeteorsManager(Game game, ref Texture2D theTexture)
            : base(game)
        {
            meteorTexture = theTexture;//Lấy texture
            meteors = new List<Meteor>();//Đang null
            // sử dụng audio library
            audio = (AudioLibrary)
```

```
            Game.Services.GetService(typeof(AudioLibrary));

        }
```

Hàm Thiết lập:

```
public override void Initialize()

        {
            meteors.Clear();//clear all meteor

            Start();
```

Vòng for thiết lập cho tất cả Meteor

```
            for (int i = 0; i < meteors.Count; i++)
            {
                meteors[i].Initialize();
            }

            base.Initialize();
        }
```

Hàm Thêm Meteor vào game, thiết lập index cho từng Meteor nhằm xác định đc các Meteor một cách riêng rẽ, sau đó thêm nó vào list các Meteor. Hàm trả về 1 new meteor và nó sẽ HĐ ngay lập tức.

```
private Meteor AddNewMeteor()
        {
            Meteor newMeteor = new Meteor(Game, ref meteorTexture);
            newMeteor.Initialize();
            meteors.Add(newMeteor);
            // Set the meteor identifier
            newMeteor.Index = meteors.Count - 1;

            return newMeteor;
        }
```

Hàm Start tạo ra các Meteor cho game, nó tạo ra biến đếm thời gian elapsedTime nhằm biết khi nào cần thêm meteor

```
    public void Start()

        {
            // Initialize a counter
            elapsedTime = TimeSpan.Zero;

            // Add the meteors
            for (int i = 0; i < STARTMETEORCOUNT; i++)
            {
                AddNewMeteor();
            }
        }
```

Bộ truy Cập dữ liệu Biến ALLMeteors

```
public List<Meteor> AllMeteors
{
    get { return meteors; }
}
```

Hàm dưới quen thuộc rồi, có trong lession trước. Tất nhiễn đã sử dụng phương thức mới nhưng chức năng thì như cũ.

```
private void CheckforNewMeteor(GameTime gameTime)

{
    // Add a rock each ADDMETEORTIME
    elapsedTime += gameTime.ElapsedGameTime;

    if (elapsedTime > TimeSpan.FromMilliseconds(ADDMETEORTIME))
    {
        elapsedTime -= TimeSpan.FromMilliseconds(ADDMETEORTIME);

        AddNewMeteor();
        // Play a sound for a new meteor
        audio.NewMeteor.Play();
    }
}
```

Hàm dưới Kiểm soát chương trình tạo ra Metẻor và Update cho từng Meteor trong List

```
public override void Update(GameTime gameTime)
{
    CheckforNewMeteor(gameTime);

    // Update Meteors
    for (int i = 0; i < meteors.Count; i++)
    {
        meteors[i].Update(gameTime);
    }

    base.Update(gameTime);
}

/// <summary>
/// Check if the ship collide with a meteor
/// <returns>true, if has a collision</returns>
/// </summary>
```

Hàm KT va chạm cho từng Object Meteor với HCN (của Ship) . Methods `CheckCollision(rect)`

Đã viết trong lớp của Meteor. Hàm `CheckForCollisions (thêm chữ “s”)` trả về true nếu có bất kỳ va chạm nào đc ghi nhận và restart lại meteor đã va cham.

```
public bool CheckForCollisions(Rectangle rect)
{
    for (int i = 0; i < meteors.Count; i++)
    {
        if (meteors[i].CheckCollision(rect))
        {
```

```
                    // BOM !!
                    audio.Explosion.Play();

                    // Put the meteor back to your initial position
                    meteors[i].PutinStartPosition();

                    return true;
                }
            }
            return false;
        }
```

Gọi hàm đồ hoạ cho từng Meteor, đồ hoạ lấy từ lớp Sprite, các bạn có thể xem lại.

```
        public override void Draw(GameTime gameTime)
        {
            // Draw the meteors
            for (int i = 0; i < meteors.Count; i++)
            {
                meteors[i].Draw(gameTime);
            }

            base.Draw(gameTime);
        }
    }
}
```

4.Lớp Player (Ship):Điều kiển Ship và KT Ship va chạm...

Lớp này toàn những thứ chúng ta đã biết, mình sẽ đi nhanh, rất nhanh ϰ :

```
#region Using Statements

using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

#endregion

namespace RockRainEnhanced
{
```

Phần Khai báo biến:

`playerIndex` Đc bổ sung nhằm mục đích tạo ra player 2

```
    public class Player : DrawableGameComponent
    {
        protected Texture2D texture;
        protected Rectangle spriteRectangle;
        protected Vector2 position;
        protected TimeSpan elapsedTime = TimeSpan.Zero;
        protected PlayerIndex playerIndex;
        protected SpriteBatch sBatch;
```

Biến screenBounds là HCN có các cạnh là khung cửa sổ game.

```
        protected Rectangle screenBounds;

        // Các biến ghi lại năng lượng và điểm của player
        protected int score;
        protected int power;
```

`INITIALPOWER` Sẽ thiết lập năng lượng ban đầu cho Ship

```
        private const int INITIALPOWER = 100;
```

Hàm Nhập dữ liệu:

Tham số tham chiếu `playerID` có tác dụng xác định rõ ràng đó là người chơi 1 hay là 2

Vì có thể có 2 player nên texture của 2 Ship là khác nhau, ta cần có 2 texture khác nhau từ đó cần 2 hcn Khác nhau khi khoanh vùng lấy texture từ đó cần phải nhập vào `rectangle`.

```
        public Player(Game game, ref Texture2D theTexture, PlayerIndex
playerID, Rectangle rectangle) : base(game)
        {
            texture = theTexture;
            position = new Vector2();
            playerIndex = playerID;
            // Get the current spritebatch
            sBatch = (SpriteBatch)
                Game.Services.GetService(typeof(SpriteBatch));

            // Create the source rectangle.
            // This represents where is the sprite picture in surface
            spriteRectangle = rectangle;

            screenBounds = new Rectangle(0, 0, Game.Window.ClientBounds.Width,
Game.Window.ClientBounds.Height);
        }
```

Hàm đặt Ship vào chỗ của nó, tuỳ theo đó là player1 hay 2

```
        public void Reset()
        {
    if (playerIndex == PlayerIndex.One)

            {
                position.X = screenBounds.Width/3;
            }
            else
            {
                position.X = (int) (screenBounds.Width/1.5);
            }

            position.Y = screenBounds.Height - spriteRectangle.Height;
            score = 0;
            power = INITIALPOWER;
        }
```

Đây là bộ truy cập dữ liệu cho score và power, đảm bảo cho score luôn ko âm.

```
public int Score
{
    get { return score; }
    set
    {
        if (value < 0)
        {
            score = 0;
        }
        else
        {
            score = value;
        }
    }
}


public int Power
{
    get { return power; }
    set { power = value; }
}
```

Sau 1 giây, tăng score hay giảm power 1 đơn vị. Khi chơi game nếu player hết power, họ sẽ die. Để đảm bảo trò chơi đc tiếp tục người chơi buộc phải ăn những cục PowerSource nhằm lên power để chơi típ. Sẽ có lớp khác đảm nhận game lôgic này.

```
private void UpdateShip(GameTime gameTime)
{
    // Keep the player inside the screen
    KeepInBound();

    // Update score
    elapsedTime += gameTime.ElapsedGameTime;

    if (elapsedTime > TimeSpan.FromSeconds(1))
    {
        elapsedTime -= TimeSpan.FromSeconds(1);
        score++;
        power--;
    }
}
```

Hàm kiểm tra Ship khi chạy game:score, power và control Ship.

```
public override void Update(GameTime gameTime)
{
    HandleInput(playerIndex);
    UpdateShip(gameTime);

    base.Update(gameTime);
}
```

```
        /// <summary>
        /// Get the ship position
        /// </summary>
```

Hàm Lựa chọn player mà gọi hàm Điều kiển phù hợp

```
        protected void HandleInput(PlayerIndex thePlayerIndex)
        {
            if (thePlayerIndex == PlayerIndex.One)
            {
                HandlePlayer1KeyBoard();
            }
            else
            {
                HandlePlayer2KeyBoard();
            }
        }
```

Ham sau giữ cho ship luôn trong màn hình.

```
        private void KeepInBound()
        {
            if (position.X < screenBounds.Left)
            {
                position.X = screenBounds.Left;
            }
            if (position.X > screenBounds.Width - spriteRectangle.Width)
            {
                position.X = screenBounds.Width - spriteRectangle.Width;
            }
            if (position.Y < screenBounds.Top)
            {
                position.Y = screenBounds.Top;
            }
            if (position.Y > screenBounds.Height - spriteRectangle.Height)
            {
                position.Y = screenBounds.Height - spriteRectangle.Height;
            }
        }
```

Hàm điều khiển cho người chơi 1

```
        private void HandlePlayer1KeyBoard()
        {
            KeyboardState keyboard = Keyboard.GetState();
            if (keyboard.IsKeyDown(Keys.Up))
            {
                position.Y -= 3;
            }
            if (keyboard.IsKeyDown(Keys.Down))
            {
                position.Y += 3;
            }
            if (keyboard.IsKeyDown(Keys.Left))
            {
                position.X -= 3;
```

```
            }
            if (keyboard.IsKeyDown(Keys.Right))
            {
                position.X += 3;
            }
        }
```

Hàm điều khiển cho người chơi 2

```
        private void HandlePlayer2KeyBoard()
        {
            KeyboardState keyboard = Keyboard.GetState();
            if (keyboard.IsKeyDown(Keys.W))
            {
                position.Y -= 3;
            }
            if (keyboard.IsKeyDown(Keys.S))
            {
                position.Y += 3;
            }
            if (keyboard.IsKeyDown(Keys.A))
            {
                position.X -= 3;
            }
            if (keyboard.IsKeyDown(Keys.D))
            {
                position.X += 3;
            }
        }
```

Vẽ con tàu thông qua `sBatch`:

```
        public override void Draw(GameTime gameTime)
        {
            // Get the current spritebatch
            sBatch = (SpriteBatch)
                Game.Services.GetService(typeof (SpriteBatch));

            // Draw the ship
            sBatch.Draw(texture, position, spriteRectangle, Color.White);

            base.Draw(gameTime);
        }

        /// <summary>
        /// Get the bound rectangle of ship position in screen
        /// </summary>
```

Hàm dưới quen thuộc rồi.

```
public Rectangle GetBounds()
        {
            return new Rectangle((int) position.X, (int) position.Y,
                spriteRectangle.Width, spriteRectangle.Height);
        }
```

```
    }
}
```

5.Lớp cho PowerSource-Thứ kéo dài trò chơi.

Nếu ko có thêm power, người chơi sẽ thua mà đối với họ điều này ko thú vị J .

Phần kế thừa:

```
#region Using Statements

using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RockRainEnhanced.Core;

#endregion

namespace RockRainEnhanced
{
    /// <summary>
    /// This is a game component that implements Power Source Element.
    /// </summary>
    public class PowerSource : Sprite
    {
```

Phần khai báo:

```
        protected Texture2D texture;
        protected Random random;
```

Phần nhập data

Nhập texture và kỹ thuật làm hoạt hình tương tự Meteor

```
        public PowerSource(Game game, ref Texture2D theTexture)
            : base(game, ref theTexture)
        {
            texture = theTexture;

            Frames = new List<Rectangle>();
            Rectangle frame = new Rectangle();
            frame.X = 291;
            frame.Y = 17;
            frame.Width = 14;
            frame.Height = 12;
            Frames.Add(frame);

            frame.Y = 30;
            Frames.Add(frame);

            frame.Y = 43;
            Frames.Add(frame);

            frame.Y = 57;
            Frames.Add(frame);
```

```
            frame.Y = 70;
            Frames.Add(frame);

            frame.Y = 82;
            Frames.Add(frame);

            frameDelay = 200;

            // Initialize the random number generator and put the power
source in your
            // start position
            random = new Random(GetHashCode());
            PutinStartPosition();
        }
```

Thiết lập vận tốc và vị trí cho powersource khi nó đc sinh ra

```
        public void PutinStartPosition()
        {
            position.X = random.Next(Game.Window.ClientBounds.Width -
                currentFrame.Width);
            position.Y = -10;
            Enabled = false;
        }

        public override void Update(GameTime gameTime)
        {
```

Câu lệnh dưới có tác dụng đưa lại powersoure cho player nếu họ chưa kịp ăn

```
if (position.Y >= Game.Window.ClientBounds.Height)

            {
                PutinStartPosition();
            }

            // Move
            position.Y += 1;

            base.Update(gameTime);
        }
```

Hàm KT va chạm, tất nhiên là với Ship

```
        public bool CheckCollision(Rectangle rect)
        {
            Rectangle spriterect =
                new Rectangle((int) position.X, (int) position.Y,
                currentFrame.Width, currentFrame.Height);
            return spriterect.Intersects(rect);
        }
    }
}
```

Ngoài ra cần hàm để tạo ra powersoucre, nó đặt trong mã lệnh của Màn Play chúng ta sẽ xem xét sau, hàm đồ hoạ dựa hoàn toàn vào tính kế thừa từ Sprite.

6. Lớp Score Nơi ghi lại thành tích và báo trước cái chết của player:

```
#region Using Statements

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;

#endregion

namespace RockRainEnhanced
{
    /// <summary>
    /// This is a game component that implements the Game Score.
    /// </summary>
```

Hàm khai báo

```
    public class Score : DrawableGameComponent
    {
        // chuẩn bị Spritebatch
        protected SpriteBatch spriteBatch = null;

        // nơi hiện Score , vị trí
        protected Vector2 position = new Vector2();

        // các giá trị sẽ sử dụng
        protected int value;// là score
        protected int power;

        protected readonly SpriteFont font;
        protected readonly Color fontColor;
```

Hàm nhập data: font

```
        public Score(Game game, SpriteFont font, Color fontColor)
            : base(game)
        {
            this.font = font;
            this.fontColor = fontColor;

            spriteBatch = (SpriteBatch)
                          Game.Services.GetService(typeof (SpriteBatch));
        }
```

Bộ truy cập

```
        public int Value
        {
            get { return value; }
            set { this.value = value; }
        }

        /// <summary>
        /// Power Value
```

```
        /// </summary>
        public int Power
        {
            get { return power; }
            set { power = value; }
        }

        /// <summary>
        /// Position of component in screen
        /// </summary>
        public Vector2 Position
        {
            get { return position; }
            set { position = value; }
        }
```

Hàm đồ hoạ, cái quan trọng nhất

```
        public override void Draw(GameTime gameTime)
        {
```

Câu lệnh dưới đây tạo 1 chuỗi là score: số điểm

```
            string TextToDraw = string.Format("Score: {0}", value);
```

hiện ra bóng chuỗi trên:

```
            spriteBatch.DrawString(font, TextToDraw, new Vector2(position.X +
1,
                                   position.Y + 1), Color.Black);
```

hiện ra chuỗi trên:, viết hơi ngược (vì thằng nào ra đời sau sẽ đè lên trên thằng ở trước)

```
            spriteBatch.DrawString(font, TextToDraw,
                                   new Vector2(position.X, position.Y),
                                   fontColor);
```

Tương tự việc in ra score nhưng có thêm height kiểu số thực là do phải viết power dưới score, chúng ta cần cách dòng, height chính là khoảng cách giữa 2 dòng trong font chữ đã dùng.

```
            float height = font.MeasureString(TextToDraw).Y;
            TextToDraw = string.Format("Power: {0}", power);
            // Draw the text shadow
            spriteBatch.DrawString(font, TextToDraw,
                new Vector2(position.X + 1, position.Y + 1 + height),
                Color.Black);
            // Draw the text item
            spriteBatch.DrawString(font, TextToDraw,
                new Vector2(position.X, position.Y + 1 + height),
                fontColor);

            base.Draw(gameTime);
        }
    }
}
```

C. Lớp cho Các Màn: Kế thừa từ GameScene

-Làm hoạt hình bằng code khi khởi đầu game, cũng giống như làm ảnh động, flash vậy.

-Cách 1 màn có thể hoạt động: hide và show

-1 màn cũng gần như 1 file game1.cs con con vậy.

-Hàm nhập Nội dung ở các class Gamescene trước, bây giờ sẽ đc điền, ...

Training XNA more and more !

1.Màn khi bắt đầu game gồm hoạt hình, menu và âm nhạc J :

```
#region Using Statements

using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Media;
using RockRainEnhanced.Core;

#endregion

namespace RockRainEnhanced
{
    /// <summary>
    /// This is a game component that implements the Game Start Scene.
    /// </summary>
    public class StartScene : GameScene
    {
```

Các thứ cần thiết phải khai báo:

Menu: hiển nhiên phải có

Texture background và cho các chữ đc làm hoạt hình (bằng code), audio

Hoạt hình cho các chữ Rock, Rain và enhanced bạn sẽ thấy chúng chạy qua lại và nhấp nháy (làm hoạt hình = code) khi run game. Chúng ta cần HCN bao quanh từ chữ và vị trí khởi đầu của chúng

```
        // Misc
        protected TextMenuComponent menu;
        protected readonly Texture2D elements;
        // Audio
        protected AudioLibrary audio;
        // Spritebatch
        protected SpriteBatch spriteBatch = null;
        // Gui Stuff
        protected Rectangle rockRect = new Rectangle(0, 0, 536, 131);
        protected Vector2 rockPosition;
        protected Rectangle rainRect = new Rectangle(120, 165, 517, 130);
        protected Vector2 rainPosition;
        protected Rectangle enhancedRect = new Rectangle(8, 304, 375, 144);
        protected Vector2 enhancedPosition;
        protected bool showEnhanced;
        protected TimeSpan elapsedTime = TimeSpan.Zero;
```

Hàm nhập data: game, 2 loại font cho 2 trạng thái của menu, texture background, texture cho chữ Rock Rain Enhanced.

```
        public StartScene(Game game, SpriteFont smallFont, SpriteFont
largeFont,
                          Texture2D background, Texture2D elements)
            : base(game)
        {
            this.elements = elements;
```

Ảnh nền thuộc kiểu ImageComponent nó có phương thức riêng để thể hiện, bạn xem lại hàm nhập data cho ImageComponent để hiểu đoạn mã dưới đây:

```
            Components.Add(new ImageComponent(game, background,
                                              ImageComponent.DrawMode.Center));
```

Tạo ra menu, gồm các chuỗi:

```
            string[] items = {"One Player", "Two Players", "Help", "Quit"};
```

Tạo ra thành phần mới.

```
            menu = new TextMenuComponent(game, smallFont, largeFont);
```

Nhập Nội dung menu

```
            menu.SetMenuItems(items);
```

Thêm vào mảng Components

```
            Components.Add(menu);
            spriteBatch = (SpriteBatch) Game.Services.GetService(
                                              typeof (SpriteBatch));
            audio = (AudioLibrary)
                Game.Services.GetService(typeof(AudioLibrary));
        }
```

Hàm dưới có tác dụng thể hiện màn start, thiết đặt toạ độ sẵn sàng cho việc chạy hoạt hình Rock Rain Enhanced... toạ độ cho Menu

```
        public override void Show()
        {
            audio.NewMeteor.Play();

            rockPosition.X = -1*rockRect.Width;
            rockPosition.Y = 40;
            rainPosition.X = Game.Window.ClientBounds.Width;
            rainPosition.Y = 180;
            // Put the menu centered in screen
            menu.Position = new Vector2((Game.Window.ClientBounds.Width -
                                         menu.Width)/2, 330);
```

tất cả thể hiện điều chưa có khi chữ rockrain chưa hoạt hình xong. Một hàm chứa if sẽ KT và khi hoạt hình xong các thuộc tính dưới đây sẽ đc true.

```
            menu.Visible = false;
            menu.Enabled = false;
```

```
            showEnhanced = false;

            base.Show();
        }
```

Hàm dưới đây ngược với hàm trên, nó dùng khi ta muốn qua cảnh khác.

```
        public override void Hide()
        {
            MediaPlayer.Stop();
            base.Hide();
        }
```

Bộ truy cập `SelectedMenuIndex` để lấy index hiện thời của phần menu đc chọn

```
        public int SelectedMenuIndex
        {
            get { return menu.SelectedIndex; }
        }
```

Hàm Update()

-Chạy hoạt hình => mở menu => chữ enhanced nhấp nháy theo thời gian.

Hoạt hình rất đơn giản, chúng ta có thể dùng các phần mềm đồ hoạ để vẽ ảnh động chả hạn, ráp vô... cũng làm đc, tuy nhiên cái ji code cứ code, coding chính xác hơn nhiều...

Kỹ thuật dưới đây tuy đơn giản nhưng rất hữu dụng.

```
        public override void Update(GameTime gameTime)
        {
            if (!menu.Visible)
            {
                if (rainPosition.X >= (Game.Window.ClientBounds.Width -
595)/2)
                {
                    rainPosition.X -= 15;
                }
```

//Chữ rain chạy từ phải qua trái

```
                if (rockPosition.X <= (Game.Window.ClientBounds.Width -
715)/2)
                {
                    rockPosition.X += 15;
                }
```

//chữ rock chạy từ trái qua phải

```
                else
                {
                    menu.Visible = true;
                    menu.Enabled = true;

                    MediaPlayer.Play(audio.StartMusic);
```

//Nổi nhạc lên

```
                    enhancedPosition = new Vector2((rainPosition.X +
                    rainRect.Width - enhancedRect.Width/2) - 80,
                    rainPosition.Y);
                      showEnhanced = true;
                }
            }
            else
            {
```

//Sau 1 s chữ enhanced hiện ra và mất đi.

```
                elapsedTime += gameTime.ElapsedGameTime;

                if (elapsedTime > TimeSpan.FromSeconds(1))
                {
                    elapsedTime -= TimeSpan.FromSeconds(1);
                    showEnhanced = !showEnhanced;
                }
            }

            base.Update(gameTime);
        }
```

Hàm Đồ hoạ:

Menu: đã có class của nó làm việc này.

Chúng ta chỉ vẽ các chữ tiêu đề game lên screen.

```
        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);

            spriteBatch.Draw(elements, rockPosition, rockRect, Color.White);
            spriteBatch.Draw(elements, rainPosition, rainRect, Color.White);
            if (showEnhanced)
            {
                spriteBatch.Draw(elements, enhancedPosition, enhancedRect,
                                 Color.White);
            }
        }
```

2.Cảnh (màn) giúp đỡ:

Cái này đơn giản hết mức, chúng ta thêm vào component của game , ảnh nền Background và image front (giúp người chơi biết cách control, đó là Ảnh của cái GamePad của XBOX và KeyBoard của PC.)

```
#region Using Statements

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using RockRainEnhanced.Core;

#endregion
```

```
namespace RockRainEnhanced
{
    /// <summary>
    /// This is a game component thats represents the Instrucions Scene
    /// </summary>
    public class HelpScene : GameScene
    {
        public HelpScene(Game game, Texture2D textureBack, Texture2D
textureFront)
            : base(game)
        {
//Ảnh nền, chế độ tô...
            Components.Add(new ImageComponent(game, textureBack,
                ImageComponent.DrawMode.Stretch));
//Ảnh help, chế độ tô...

            Components.Add(new ImageComponent(game, textureFront,
                ImageComponent.DrawMode.Center));
        }
    }
```

Trong file mẫu, front image không có vùng trống để background thể hiện nên add background vào cũng không nhận thấy đc.

3. Màn Action – nơi play game

Chúng ta sẽ có nhiều việc để thảo luận đây, thằng này với Game1.cs nữa là xong Phần nâng cấp project, gắng lên nào !

```
using System;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Media;
using RockRainEnhanced.Core;
```

```
namespace RockRainEnhanced
{
    /// <summary>
    /// This is a game component that implements the Action Scene.
    /// </summary>
    public class ActionScene : GameScene
    {
```

Khai báo biến:

Texture `actionTexture` chứa các thành phần để play game (ship, meteor, power...) một texture nữa là background nhưng chúng ta xếp nó vào ImageComponent, thứ đã có class.

Khi nhấn nút Ênter, game sẽ pause (để cho player làm việc bận của họ), nhấn Enter sau đó, game sẽ resume lại và cuộc chơi tiếp tục, khi đó trên màn hình sẽ có chữ “pause”. Khi người chơi thua (vì họ có chơi cả đời cũng ko win đc, chỉ điểm cao hơn thôi) sẽ có chữ “game over”, trong dự án này 2 chữ đó là 2 cái texture, do đó nó cần có vị trí và HCN lấy texture khi draw: `pausePosition ,gameoverPosition`

```
        protected Texture2D actionTexture;
        private AudioLibrary audio;
        protected SpriteBatch spriteBatch = null;

        // Game Elements
        protected Player player1;
        protected Player player2;
        protected MeteorsManager meteors;
        protected PowerSource powerSource;
        protected ImageComponent background;
        protected Score scorePlayer1;
        protected Score scorePlayer2;

        // Gui Stuff
        protected Vector2 pausePosition;
        protected Vector2 gameoverPosition;
        protected Rectangle pauseRect = new Rectangle(1, 120, 200, 44);
        protected Rectangle gameoverRect = new Rectangle(1, 170, 350, 48);
```

2 biến boolean KT xem có pause hay over ko là hiển nhiên phải có.

Tạo bộ đếm time và KT xem có phải 2 người chơi hay ko

```
        protected bool paused;
        protected bool gameOver;
        protected TimeSpan elapsedTime = TimeSpan.Zero;
        protected bool twoPlayers;
```

Hàm nhập data:

-nhập những thứ của 1 GameSnece thứ thiệt.

-Cần 1 imageComponent làm Background

-Sử dụng texture, SB và Audio

```
public ActionScene(Game game, Texture2D theTexture,
    Texture2D backgroundTexture, SpriteFont font)
    : base(game)
{
    background = new ImageComponent(game, backgroundTexture,
        ImageComponent.DrawMode.Stretch);
    Components.Add(background);

    actionTexture = theTexture;

    // Get the current sprite batch
    spriteBatch = (SpriteBatch)
        Game.Services.GetService(typeof (SpriteBatch));

    // Get the audio library
    audio = (AudioLibrary)
        Game.Services.GetService(typeof(AudioLibrary));
```

Tạo 1 bộ quản lý Meteor, từ đó tạo 1 đàn Meteor

```
    meteors = new MeteorsManager(Game, ref actionTexture);
    Components.Add(meteors);
```

Chuẩn bị sẵn 2 player

```
    player1 = new Player(Game, ref actionTexture, PlayerIndex.One,
        new Rectangle(323, 15, 30, 30));
    player1.Initialize();
    Components.Add(player1);

    player2 = new Player(Game, ref actionTexture, PlayerIndex.Two,
        new Rectangle(360, 17, 30, 30));
    player2.Initialize();
    Components.Add(player2);
```

Chuẩn bị sẵn 2 Score ghi điểm cho 2 player

```
    scorePlayer1 = new Score(game, font, Color.Blue);
    scorePlayer1.Position = new Vector2(10, 10);
    Components.Add(scorePlayer1);
    scorePlayer2 = new Score(game, font, Color.Red);
    scorePlayer2.Position = new Vector2(
        Game.Window.ClientBounds.Width - 200, 10);
    Components.Add(scorePlayer2);
```

Tạo ra power Source và thiết lập cho nó theo class của nó mà chúng ta đã viết.

```
    powerSource = new PowerSource(game, ref actionTexture);
    powerSource.Initialize();
    Components.Add(powerSource);
}
```

Hàm khi Hiển thị cảnh:

```
public override void Show()
{
    MediaPlayer.Play(audio.BackMusic);
```

Khởi tạo cho bộ quản lý các meteor (MeteorsManager), tự nó tạo ra các meteor

```
            meteors.Initialize();
            powerSource.PutinStartPosition();
```

khởi động cho 2 player

```
            player1.Reset();
            player2.Reset();
```

Hiển thị chữ “pause” hoặc “game over” khi cần, mặc định là không.

```
            paused = false;
            pausePosition.X = (Game.Window.ClientBounds.Width -
                pauseRect.Width)/2;
            pausePosition.Y = (Game.Window.ClientBounds.Height -
                pauseRect.Height)/2;

            gameOver = false;
            gameoverPosition.X = (Game.Window.ClientBounds.Width -
                gameoverRect.Width)/2;
            gameoverPosition.Y = (Game.Window.ClientBounds.Height -
                gameoverRect.Height)/2;
```

Player 2 có hiển thị và HĐ hay ko tuỳ thuộc biến twoPlayer

```
            player1.Visible = true;
            player2.Visible = twoPlayers;
            player2.Enabled = twoPlayers;
            scorePlayer2.Visible = twoPlayers;
            scorePlayer2.Enabled = twoPlayers;

            base.Show();
        }
```

Hàm thực hiện khi ẩn scene

```
        public override void Hide()
        {
            // Stop the background music
            MediaPlayer.Stop();
            base.Hide();
        }
```

Các bộ truy cập cho các biến điều khiển quan trọng

```
        public bool TwoPlayers
        {
            get { return twoPlayers; }
            set { twoPlayers = value; }
        }

        public bool GameOver
        {
            get { return gameOver; }
        }

        public bool Paused
        {
            get { return paused; }
```

```
            set
            {
                paused = value;
//nhận thông tin từ biến paused
                if (paused)
                {
//Dừng chơi nhạc khi “pause”
                    MediaPlayer.Pause();
                }
                else
                {
//Sau khi hết pause, âm nhạc trở lại
                    MediaPlayer.Resume();
                }
            }
        }
```

Hàm gây thiệt hại cho player

Player 1 chắc chắn phải có, player 2 có hay ko còn tuỳ, dùng lệnh if để KT.

```
        private void HandleDamages()
        {
            // Check Collision for player 1
            if (meteors.CheckForCollisions(player1.GetBounds()))
            {
                // Player penalty
                player1.Power -= 10;
                player1.Score -= 10;
            }

            // Check Collision for player 2
            if (twoPlayers)
            {
                if (meteors.CheckForCollisions(player2.GetBounds()))
                {
                    // Player penalty
                    player2.Power -= 10;
                    player2.Score -= 10;
                }
```

Nếu 2 player tông vào nhau cả 2 đều bị thiệt hại.

```
                if (player1.GetBounds().Intersects(player2.GetBounds()))
                {

                    player1.Power -= 10;
                    player1.Score -= 10;

                    player2.Power -= 10;
                    player2.Score -= 10;
                }
            }
        }
```

Hàm nhận năng lượng từ powersource

```
        private void HandlePowerSourceSprite(GameTime gameTime)
        {
            // Player 1 get the power source
            if (powerSource.CheckCollision(player1.GetBounds()))
            {
```

Chơi nhạc

```
                audio.PowerGet.Play();
```

Đặt lại bộ đếm time, sẵn sàng cho power tiếp theo

```
                elapsedTime = TimeSpan.Zero;
```

Đặt lại power vào vị trí

```
                powerSource.PutinStartPosition();
```

Tăng thêm cho player 50 power

```
                player1.Power += 50;
            }
```

Nếu có playêr2 , cũng tương tự player1

```
            if (twoPlayers)
            {
                // Player 2 get the power source
                if (powerSource.CheckCollision(player2.GetBounds()))
                {
                    audio.PowerGet.Play();
                    elapsedTime = TimeSpan.Zero;
                    powerSource.PutinStartPosition();
                    player2.Power += 50;
                }
            }
```

Bộ đếm time tạo ra power trong 15s

```
            elapsedTime += gameTime.ElapsedGameTime;
            if (elapsedTime > TimeSpan.FromSeconds(15))
            {
                elapsedTime -= TimeSpan.FromSeconds(15);
                powerSource.Enabled = true;
                audio.PowerShow.Play();
            }
        }

        /// <summary>
        /// Allows the game component to update itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>
        public override void Update(GameTime gameTime)
        {
            if ((!paused) && (!gameOver))
            {
                // Check collisions with meteors
                HandleDamages();

                // Check if a player get a power boost
                HandlePowerSourceSprite(gameTime);
```

Bộ truy cập dữ liệu ở lớp Score sẽ truyền điểm của playêr vào cái hàm drraw của lớp score và do đó, điểm số sẽ đc Update liên tục, tương tự cho power

```
            scorePlayer1.Value = player1.Score;
            scorePlayer1.Power = player1.Power;
            if (twoPlayers)
            {
                scorePlayer2.Value = player2.Score;
                scorePlayer2.Power = player2.Power;
            }
```

Trò chơi kết thúc khi player1 hoặc 2 dead

```
            gameOver = ((player1.Power <= 0) || (player2.Power <= 0));
            if (gameOver)
            {
                player1.Visible = (player1.Power > 0);
                player2.Visible = (player2.Power > 0) && twoPlayers;
                // Stop the music
                MediaPlayer.Stop();
            }
```

Lệnh dươí đây chạy hàm update cho toàn bộ component

```
            base.Update(gameTime);
        }
```

Nếu player Thua, meteor vẫn hoạt động đề phòng họ cay cú và muốn restart, tất nhiên vân bắt đầu từ cảnh start đến action

```
            if (gameOver)
            {
                meteors.Update(gameTime);
            }
        }

        /// <summary>
        /// Allows the game component to draw itself.
        /// </summary>
        /// <param name="gameTime">Provides a snapshot of timing
values.</param>
        public override void Draw(GameTime gameTime)
        {
```

Các component khác tự nó draw đc trừ chữ gameover và pause

```
            base.Draw(gameTime);
```

Hiện chữ pause

```
            if (paused)
            {
                // Draw the "pause" text
                spriteBatch.Draw(actionTexture, pausePosition, pauseRect,
                    Color.White);
            }
```

Hiện chữ game over

```
            if (gameOver)
            {
                // Draw the "gameover" text
                spriteBatch.Draw(actionTexture, gameoverPosition,
gameoverRect,
                    Color.White);
            }
}
    }
}
```

Một chặng đường dài, Game1.cs là tia sáng cuối đường hầm.

D.Lớp Game1.cs

Cái Dự án này dài lê thê, bạn có nhận thấy ko, tất cả những j ta viết sẽ đc quản lý bởi game1.cs nó là linh hồn của bất cứ game nào viết từ XNA.

Khai báo toàn bộ những résource trong game đồng thời bổ sung theo các hàm điều khiển trong từng màn (cái này chúng ta chưa viết đúng ko nào), đồng thời quy định trình tự xuất hiện các màn, còn ở màn nào xảy ra cái j thì chúng ta đã viết trong class của màn (scene) của nó rồi.

Chúng ta ko sử dụng gì khác lạ, tất cả vẫn rất quen thuộc

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using RockRainEnhanced.Core;

namespace RockRainEnhanced
{
    public class Game1 : Game
```

Phần khai báo :

```
    {
        private readonly GraphicsDeviceManager graphics;
        private SpriteBatch spriteBatch;
```

Các texture làm nền cho từng màn (game1 là lớp chạy game, tất cả resource đều phải tải.)

```
        protected Texture2D helpBackgroundTexture, helpForegroundTexture;
        protected Texture2D startBackgroundTexture, startElementsTexture;
        protected Texture2D actionElementsTexture, actionBackgroundTexture;
```

Cac màn của game, mỗi màn ta cứ coi như là 1 game1.cs nhỏ nhỏ xinh xinh cũng đc 3 màn đã viết class và 1 màn chung activeScene , nó đc gán cho màn hiện hành nhằm xác định chúng ta đang ở màn nào (VD: nếu ở màn help thì ActiveScene = helpScene)

```
        protected HelpScene helpScene;
```

```
        protected StartScene startScene;
        protected ActionScene actionScene;
        protected GameScene activeScene;
```

// Audio Stuff

```
        private AudioLibrary audio;
```

// Fonts

```
        private SpriteFont smallFont, largeFont, scoreFont;
```

//Thiết bị đầu vô: bàn phím, nó có chữ "old" tức là bàn phím sử dụng trước, do sử dụng trước nên nó mới "cũ", nếu bạn có trí nhớ tốt bạn sẽ nhớ đến kỹ thuật cài đặt sự kiện release một phím.

```
        protected KeyboardState oldKeyboardState;
```

Hàm nhập data

```
        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
```

// Khởi tạo cho bàn phím

```
            oldKeyboardState = Keyboard.GetState();         }



        protected override void LoadContent()
        {
```

Những thứ như thường lệ

```
            spriteBatch = new SpriteBatch(GraphicsDevice);
            Services.AddService(typeof (SpriteBatch), spriteBatch);
```

Thiết lập cho object quản lý âm thanh

```
            audio = new AudioLibrary();
            audio.LoadContent(Content);
            Services.AddService(typeof(AudioLibrary), audio);
```

// gán 2 texture cho 2 biến ở cảnh help

```
            helpBackgroundTexture =
Content.Load<Texture2D>("helpbackground");
            helpForegroundTexture =
Content.Load<Texture2D>("helpForeground");
```

Nhập thông tin cho màn help, sau đó add nó vào mảng components

```
            helpScene = new HelpScene(this, helpBackgroundTexture,
                    helpForegroundTexture);
            Components.Add(helpScene);
```

Load các thành phần đồ hoạ cho màn start

```
            smallFont = Content.Load<SpriteFont>("menuSmall");
            largeFont = Content.Load<SpriteFont>("menuLarge");
            startBackgroundTexture =
Content.Load<Texture2D>("startbackground");
            startElementsTexture =
Content.Load<Texture2D>("startSceneElements");
```

Nhập info cho màn start

```
            startScene = new StartScene(this, smallFont, largeFont,
                startBackgroundTexture, startElementsTexture);
            Components.Add(startScene);
```

Tương tự, load và thiết lập cho màn chơi (action)

actionElementsTexture = Content.Load<Texture2D>("rockrainenhanced");

```
            actionBackgroundTexture =
Content.Load<Texture2D>("SpaceBackground");
            scoreFont = Content.Load<SpriteFont>("score");
            actionScene = new ActionScene(this, actionElementsTexture,
                actionBackgroundTexture, scoreFont);
            Components.Add(actionScene);
```

Màn nào sẽ bắt đầu thể hiện, cái này quá rõ rồi J

```
            startScene.Show();
            activeScene = startScene;
        }
```

Hàm điều khiển việc show một scene:

`activeScene.Hide(); activeScene` Kế thừa từ GameScene thế nên nó cũng là một scene thứ thiết (hiện tại là null tức là chả có j trong cái active đó hết) thế nên chúng ta cũng nên hide nó để màn gán cho active màn muốn show ra ngoài, Sau đó cho màn đc thể hiện đc gán cho activeScene và show nó. Đó là khi khai báo, còn nếu chạy chương trình thức sự, active đc hiểu là màn trước (cái chúng ta đã sử dụng đã dùng chán (màn help khi đã biết, hiểu hết luật, màn action khi gameover...) còn tham số tham chiếu scene đc hiểu là màn chúng ta muốn tới (màn action là màn chúng ta muốn tới sau khi dã coi qua màn help). Tóm lại để chuyển từ màn active đến màn scene chúng ta sử dụng hàm sau:

```
        protected void ShowScene(GameScene scene)
        {
            activeScene.Hide();
            activeScene = scene;
            scene.Show();
        }
```

cái này thể hiện tính đóng gói, việc chuyển màn đơn giản chỉ là:

ShowScene(Gamescene tên màn);

Hàm Update game, ...

```
        protected override void Update(GameTime gameTime)
        {
            // lấy input phù hợp
            HandleScenesInput();
//mỗi component đều có hàm update của riêng nó...

            base.Update(gameTime);
        }
```

Khi nhấn Enter, chúng ta sẽ dừng game lại tạm thời cho đến khi nút enter đc nhấn lần thứ 2, để biết khi nào đã pause hay ko, hàm dưới đây thực hiện điều đó, nó trả về true nếu đang pause và false nếu ngược lại.

```
private bool CheckEnterA()
        {

            KeyboardState keyboardState = Keyboard.GetState();
```

Release phím ẻnter mới có hiệu lực

```
            bool result = (oldKeyboardState.IsKeyDown(Keys.Enter) &&
                (keyboardState.IsKeyUp(Keys.Enter)));

            oldKeyboardState = keyboardState;
            return result;
        }
```

Mỗi class cho từng cảnh, chúng ta chưa viết hàm input điều khiển bằng bàn phím để chuyển màn cho từng thằng. hàm dưới đây kiểm tra màn đang chơi là màn gì mà lấy input() cho phù hợp

```
 private void HandleScenesInput()
        {
            // Handle Start Scene Input
            if (activeScene == startScene)
            {
                HandleStartSceneInput();
            }
            // Handle Help Scene input
            else if (activeScene == helpScene)
            {
                if (CheckEnterA())
                {
                    ShowScene(startScene);
                }
            }
            // Handle Action Scene Input
            else if (activeScene == actionScene)
            {
                HandleActionInput();
```

```
        }
    }
```

ActionScene thiếu hàm chuyên màn, chúng ta thêm nó vào:

```
    private void HandleActionInput()
    {
        KeyboardState keyboardState = Keyboard.GetState();

        bool backKey = (oldKeyboardState.IsKeyDown(Keys.Escape) &&
            (keyboardState.IsKeyUp(Keys.Escape)));
```

//Khi release phim Esc

```
        bool enterKey = (oldKeyboardState.IsKeyDown(Keys.Enter) &&
            (keyboardState.IsKeyUp(Keys.Enter)));
```

//Khi release phím enter

```
        oldKeyboardState = keyboardState;
```

// khi gameover nhấn Enter chúng ta nhảy ra màn start

```
        if (enterKey)
        {
            if (actionScene.GameOver)
            {
                ShowScene(startScene);
            }
            Else
```

//Toán tử ! tức là gán ngược lại giá trị boolean, nếu pause đang false thì nó trở về true và ngược lại đồng thời âm nhạc vang lại lên, nếu pause thì nhạc nó tự tắt, kỹ thuật resume này cần bổ sung play cho nhạc nền

```
            {
                audio.MenuBack.Play();
                actionScene.Paused = !actionScene.Paused;
            }
        }

        if (backKey)
        {
            ShowScene(startScene);
        }
    }
```

Lưu ý các bạn một tí để tránh nhầm lẫn: biến `keyboardState` các bạn thấy nó cũng ở hàm CheckEnterA() đúng không, và 2 biến này là 2 hay cũng chỉ là 1? Chúng là 2 biến khác nhau hoàn toàn bởi lẽ chúng đc khai báo trong từng hàm => đây là biến địa phương và chỉ hoạt động trong phạm vi của hàm đó, nếu đem ra ngoài Class của Game1, trình biên dịch sẽ ko hiểu đc

```
    /// <summary>
    /// Handle buttons and keyboard in StartScene
    /// </summary>
```

Tương tự cho mở màn (startScene)

```
        private void HandleStartSceneInput()
        {
            if (CheckEnterA())
            {
```

//tuỳ theo người chơi nhấn Enter ở index nào của menu mà đưa họ tới màn phù hợp

Biến `actionScene.TwoPlayers` có nghĩa là actionScene là gốc, twoPlayer và biến thuộc màn này và nó dưới quyền của Action do đó, dấu chấm "." Thể hiện điều này

```
                audio.MenuSelect.Play();
                switch (startScene.SelectedMenuIndex)
                {
                    case 0:
                        actionScene.TwoPlayers = false;
                        ShowScene(actionScene);
                        break;
                    case 1:
                        actionScene.TwoPlayers = true;
                        ShowScene(actionScene);
                        break;
                    case 2:
                        ShowScene(helpScene);
                        break;
                    case 3:
                        Exit();
                        break;
                }
            }
        }
```

Hàm draw đơn giản, cơ bản và nó gọi hàm draw của tất cả của draw của các component, OOP là thế đấy :D

```
                protected override void Draw(GameTime gameTime)
        {
            // Begin..
            spriteBatch.Begin();

            // Draw all Game Components..
            base.Draw(gameTime);

            // End.
            spriteBatch.End();
        }
    }
}
```

Đã xong, một game cơ bản cần những thứ như vậy. Bài viết tạm end ở đây,... một tuần vất vả

See you again! `J`