Clase 20

Jueves, 19 Octubre 2023

Indice

Interfaz gráfica

- Es posible crear objetos de forma dinámica. Para ello utilizamos un bucle 'for' para crear un numero 'i' de personajes

```
'
for(int i = 0; i<countEnemies; i++)
{
    Character character = new Character();
    list.Add(character);
}
'
```

- Estudio del color. RGB frente a CMYK.
  RGB es habitual utilizarlo cuando pintamos sobre fondos negros, pantallas.
  CMYK es habitual utilizarlo cuando pintamos sobre fondos blancos, papel. Lo normal es necesitar imprimir el color negro cuando interactuamos sobre pantallas blancas.
- Es mucho mas util para un programador pensar en tantos por uno.
- Para pintar todos los elementos de nuestra pantalla, utilizamos las listas.
- Existen dos formas de utilizar los parametros de las funciones:
  ' r = 1.0; g = 1.0; b = 1.0; Draw(r,g,b); '
  En el momento de dibujar recogemos las variables
  ' Draw(r = 1.0, g = 1.0, b = 1.0); '
  Primero asigna todos los valores, y después realiza la función Draw(); Es menos aconsejable.
- El **Draw()** no cambia valores ni posiciones. El **Draw()** unicamente dibuja. Tenemos el **Animate()** para realizar cambios en el dibujo.
- Si tenemos valores que son muy repetitivos y largos de escribir, lo mas conveniente es utilizar variables donde almacenar estos valores
  ' player[i] // Mejor en una variable Character pj; pj = player[i]; '
- Estudiamos la filosofía de la programación orientada a objetos.
  Vamos a cambiar el funcionamiento de la función **OnDraw()**.

```
'
public class World
{
    public void OnDraw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            canvas.FillShader.SetColor(r,g,b,a);
            canvas.DrawRectangle(x,y,w,h);
        }
    }
}
'


Por este otro, mejor orientado a los objetos.


'
public class World
{
    public void OnDraw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            list[i].Draw(canvas);
        }
    }
}
```

```
public class Character
{
    public void Draw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            canvas.FillShader.SetColor(r,g,b,a);
            canvas.DrawRectangle(x,y,w,h);
        }
    }
}
'
```

Aquí se ordena al elemento 'i' de la lista que tiene la clase 'Mundo', que se pinte. Esta orden invoca a la función de objeto de la clase Character, donde se establece como debe pintarse este personaje.
Hemos separado el funcionamiento de Draw(), de manera que ahora desde la clase superior, llamamos a la funcion del objeto instancia para que se pinte.
Es importante pasarle el 'Icanvas canvas' para que pueda utilizar la función del SDK.

- Siempre que utilizamos metodos de objeto y llamadas de ese objeto, se añade implicitamente una orden 'this.', que es invisible para el usuario gracias al runtime, y que apunta hacia el objeto que se encuentra en ese mismo momento en ejecución.

```
'
public class Character
{
    public void Draw(Icanvas canvas)
    {
        for(int i = 0; i<list.Count; i++)
        {
            canvas.FillShader.SetColor(this.r,this.g,this.b,this.a);
            canvas.DrawRectangle(this.x,this.y,this.w,this.h);
        }
    }
'
```

- A partir de ahora vamos a separar las clases según su jerarquía en el programa. Cada clase crea un objeto de una clase mas específica, de modo que el funcionamiento del programa sea, a la vez eficiente y sencillo de entender.
- Debemos crear las siguientes clases en nuestro programa: Program, MyGame, World, Character y Utils
- Las Listas, realmente son arrays con Metodos diseñados por los programadores de C# para facilitar su uso.
- El Program perfecto es el siguiente:

```
'
public class Program
{
    public static void Main()
    {
        MyGame game = new MyGame();
        UDK.Game.Launch(game);
    }
}
'
```