Live coding serpiente

El Post de hoy consiste en un juego muy común que todos conocemos de los antiguos Nokia.

La idea de este ejercicio es ver o trabajar la gran mayoría de elementos y técnicas de programación que hemos estado viendo a lo largo del **curso de programación básica**.

Índice

El Juego de la serpiente o snake

En este post veremos una solución a como podemos realizar este juego en .net

1 - Análisis

Lo primero de todo realizaremos un análisis de los requisitos del juego que son a groso modo:

- Tenemos un tablero o un terreno donde podemos mover la serpiente
- La serpiente se puede mover
  - Arriba
  - Abajo
  - Derecha
  - Izquierda
- Caramelos que salen en la pantalla para comerselos y sumar puntos

Por lo tanto si convertimos las entidades reales a entidades de código, **como vimos en el video de objetos y clases**, obtendremos lo siguiente:

. Clase Tablero
- Propiedades:
  i. Altura
  ii. Anchura
- Métodos:
  i. Dibujar el tablero

. Clase serpiente
- Propiedades:
  i. Cola = Lista de posiciones
  ii. Dirección actual
  iii. Puntos

- Métodos
 i. Morir
 ii. Moverse
iii. Comerse el caramelo
. Clase Caramelo
- Aparece

Además de estas clases, quizá debamos crear alguna adicional, como por ejemplo una clase `Util` que nos dibuje por pantalla una línea o caracter, ya que es una acción que vamos a repetir constantemente.

```
static class Util
{
    public static void DibujarPosicion(int x, int y, string caracter)
    {
        Console.SetCursorPosition(x, y);
        Console.WriteLine(caracter);
    }
}
```

Como podemos obsevar el método `Console.SetCursorPosition(x,y)` nos ubica el cursor dentro de la terminal en el punto que deseamos.

Y como vimos en el video de **entrada salida por teclado y pantalla**, Console.WriteLine(string) nos escribe un texto, en la gran mayoría de casos en este ejercicio, es tan solo un carácter. En el caso de los muros es el caracter `#` o la `x` para la serpiente

2 - Código

Cuando programamos debemos pensar como si tuvieramos ese objeto, u objetos, delante, por lo que el primer elemento que pondríamos sería el tablero, así que la entidad Tablero será la primera que vayamos a crear.

2.1 - Entidad tablero

Creamos la clase tablero con las propiedades `Altura` y `Anchura`, además, ya que la utilizaremos más adelante, creamos la propiedad `ContieneCaramelo` la cual la inicializamos a `false`, ya que cuando dibujamos el tablero, de primera instancia, el caramelo no existe.

También debemos crear el método DibujarTablero el cual recorre la altura y la anchura para dibujar el tablero indicado.

```
public class Tablero
{
```

```
    public readonly int Altura;
    public readonly int Anchura;
    public bool ContieneCaramelo;
    public Tablero(int altura, int anchura)
    {
        Altura = altura;
        Anchura = anchura;
        ContieneCaramelo = false;
    }

    public void DibujarTablero()
    {
        for (int i = 0; i <= Altura; i++)
        {
            Util.DibujarPosicion(Anchura, i, "#");
            Util.DibujarPosicion(0, i, "#");
        }

        for (int i = 0; i <= Anchura; i++)
        {
            Util.DibujarPosicion(i, 0, "#");
            Util.DibujarPosicion(i, Altura, "#");
        }
    }
}
```

2.2 - Entidad Serpiente

Por supuesto es muy importante que creemos la entidad serpiente. Para ello es importante que recordemos los datos anteriores. Como hemos dicho, contendrá una lista de posiciones, por lo que `Posicion` en si misma será una entidad. La cual contendrá la posición sobre el eje de las X y la posición sobre el eje de las Y.

```
public class Posicion
{
    public  int X;
    public  int Y;

    public Posicion(int x, int y)
    {
        X = x;
        Y = y;
    }
}
```

Además de `List<Posicion>` deberá contener la direccion en la que se va a mover. Estas al ser solamente 4 y ser siempre las mismas, las crearemos en una enumeración

```
public enum Direccion
{
    Arriba,
    Abajo,
    Izquierda,
    Derecha
}
```

Utilizando la enumeración nos resulta mucho mas fácil saber que dirección estamos apuntando.

Finalmente la cla

```
public class Serpiente
{
    List<Posicion> Cola { get; set; }
    public Direccion Direccion { get; set; }
    public int Puntos { get; set; }

    public Serpiente(int x, int y)
    {

        Posicion posicionInicial = new Posicion(x, y);
        Cola = new List<Posicion>() { posicionInicial };
        Direccion = Direccion.Abajo;
        Puntos = 0;
    }

    public void DibujarSerpiente()
    {
        foreach (Posicion posicion in Cola)
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Util.DibujarPosicion(posicion.X, posicion.Y, "x");
            Console.ResetColor();
        }
    }

    public bool ComprobarMorir(Tablero tablero)
    {
       throw new NotImplementedException();
    }
```

```
    public void Moverse(bool haComido)
    {
       throw new NotImplementedException();
    }

    public bool ComeCaramelo(Caramelo caramelo, Tablero tablero)
    {
        throw new NotImplementedException();
    }
}
```

Fin ón
y d la
cua

Cor la
col El
cód a.
Mie

El r

2.3
No do
nos os
la e

```
public class Caramelo
{
    public Posicion Posicion { get; set; }

    public Caramelo(int x, int y)
    {
        Posicion = new Posicion(x, y);
    }

    public void DibujarCaramelo()
    {
        Console.ForegroundColor = ConsoleColor.Blue;
        Util.DibujarPosicion(Posicion.X, Posicion.Y, "O");
        Console.ResetColor();
    }

    public static Caramelo CrearCaramelo(Serpiente serpiente, Tablero tablero)
```

```
    {
        throw new NotImplementedException();
    }
}
```

3 - I

Ahc in
imp y
vay

Par ue
se

Par no
incl la
pro ue
se

```
static void Main(string[] args)
{
    Tablero tablero = new Tablero(20, 20);

    Serpiente serpiente = new Serpiente(10, 10);
    Caramelo caramelo = new Caramelo(0, 0);
    bool haComido = false;

    do
    {
        Console.Clear();
        tablero.DibujarTablero();
        serpiente.Moverse(haComido);
        haComido = serpiente.ComeCaramelo(caramelo, tablero);
        serpiente.DibujarSerpiente();



    } while (serpiente.ComprobarMorir(tablero));


    Console.ReadKey();
}
```

Debemos inicializar tanto el `tablero` como la `serpiente` como el primer `caramelo` fuera del bucle, ya que si está dentro del mismo estaríamos empezando desde 0 todo el rato.

Además, la primera línea dentro del bucle debe ser para limpiar la consola, utilizando un `Console.Clear()` ya que esta no se limpia automáticamente.

Finalmente, de esta pequeña parte de código, debemos recordar que la serpiente primero se mueve y luego come. Finalizando con dibujarla.

3.1 - Serpiente se mueve

El primer paso del programa es moverse, para ello deberemos modificar la posición en la que nos encontramos, dependiendo de la dirección:

- **Arriba**: Y -1
- **Abajo**: Y+1
- **Derecha**: X +1
- **Izquierda**: X-1

Obtenemos la primera posición de la cola (osea la cabeza) y actualizamos su valor como podemos ver en `ObtenerNuevaPrimeraPosicion()` .

```
public void Moverse(bool haComido)
{
    List<Posicion> nuevaCola = new List<Posicion>();
    nuevaCola.Add(ObtenerNuevaPrimeraPosicion());
    nuevaCola.AddRange(Cola);

    if (!haComido)
    {
        nuevaCola.Remove(nuevaCola.Last());
    }

    Cola = nuevaCola;
}

private Posicion ObtenerNuevaPrimeraPosicion()
{
    int x = Cola.First().X;
    int y = Cola.First().Y;

    switch (Direccion)
    {
        case Direccion.Abajo:
            y += 1;
            break;
        case Direccion.Arriba:
            y -= 1;
```

```
                break;
            case Direccion.Derecha:
                x += 1;
                break;
            case Direccion.Izquierda:
                x -= 1;
                break;
        }
        return new Posicion(x, y);
    }
```

Par un
val

Ent na
list

Por

Po
10
10
10

El el
sig

Po
10

Par

Po
10
10
10
10

Per en
el l na
con

Cor

Posición X	Posición Y
10	9
10	10
10	11

Como observamos son las nuevas posiciones de la serpiente.

3.2 - Serpiente come

Para comprobar si la serpiente ha comido, debemos comprobar que el caramelo no esta en ninguna posición de la cola. No debería pasar que este se cree en una posición de la cola, pero por si acaso las comprobamos todas.

Dentro de la clase serpiente el código para comprobar si hemos comido es el siguiente:

```
public bool ComeCaramelo(Caramelo caramelo, Tablero tablero)
{
    if (PosicionEnCola(caramelo.Posicion.X, caramelo.Posicion.Y))
    {
        Puntos += 10; // sumamos puntos
        tablero.ContieneCaramelo = false;//Quitar el caramelo o generar uno nuevo
        return true;
    }
    return false;

}

public bool PosicionEnCola(int x, int y)
{
    return Cola.Any(a => a.X == x && a.Y == y);
}
```

`ComeCaramelo`, devuelve `true` si nos comemos el caramelo, además indica al tablero que ya no tiene un caramelo. Lo que obligará al mismo a generar uno nuevo.

3.3 - Moverse en diferentes direcciones

Pero claro, tal y como está ahora solo nos movemos en una dirección. Lo que debemos hacer es permitir que el usuario introduzca la dirección para la que queremos que se mueva.

Para ello dentro de nuestro bucle debemos introducir el siguiente código:

```
var sw = Stopwatch.StartNew();
while (sw.ElapsedMilliseconds <= 250)
{
    serpiente.Direccion = Util.LeerMovimiento(serpiente.Direccion);
}
```

El método `LeerMovimiento` nos devolera la `Direccion` que hemos pulsado, si no pulsamos una nueva dirección durante 200milisegundos, devolvera automaticamente la dirección actual.

```
static Direccion LeerMovimiento(Direccion movimientoActual)
{
    if (Console.KeyAvailable)
    {
        var key = Console.ReadKey().Key;
```

```
        if (key == ConsoleKey.UpArrow && movimientoActual != Direccion.Abajo)
        {
            return Direccion.Arriba;
        }
        else if (key == ConsoleKey.DownArrow && movimientoActual != Direccion.Arril
        {
            return Direccion.Abajo;
        }
        else if (key == ConsoleKey.LeftArrow && movimientoActual != Direccion.Dere
        {
            return Direccion.Izquierda;
        }
        else if (key == ConsoleKey.RightArrow && movimientoActual != Direccion.Izq
        {
            return Direccion.Derecha;
        }
    }
    return movimientoActual;
}
```

Debemos evitar ir en dirección contraria a la que vamos actualente, o eso causara que choquemos y muramos contra nosotros mismos.

3.4 - Dibujar el caramelo

Dibujar el caramelo es muy sencillo, realizaremos una acción similar a cuando dibujamos la serpiente o los muros. La única diferencia es que, para dibujar el caramelo, debemos estar seguros de dos factores.

- Es totalmente aleatorio
- No se encuentra en los muros ni en la serpiente.

```
public void DibujarCaramelo()
{
    Console.ForegroundColor = ConsoleColor.Blue;
    Util.DibujarPosicion(Posicion.X, Posicion.Y, "O");
    Console.ResetColor();
}

public static Caramelo CrearCaramelo(Serpiente serpiente, Tablero tablero)
{
    bool carameloValido;
    int x,y;

    do
```

```
    {
        Random random = new Random();
        x = random.Next(1, tablero.Anchura - 1);
        y = random.Next(1, tablero.Altura - 1);
        carameloValido = serpiente.PosicionEnCola(x, y);

    } while (carameloValido);

    tablero.ContieneCaramelo = true;
    return new Caramelo(x, y);
}
```

3.5
Par

- La
- La

```
public void ComprobarMorir(Tablero tablero)
{
    //Si nos chocamos contra nosotros
    Posicion primeraPosicion = Cola.First();

    EstaViva = !((Cola.Count(a => a.X == primeraPosicion.X && a.Y == primeraPosici
        || CabezaEstaEnPared(tablero, Cola.First()));

}

//si la primera posicion esta en cualquiera de los muros, morimos.
private bool CabezaEstaEnPared(Tablero tablero, Posicion primeraPoisicon)
{
    return primeraPoisicon.Y == 0 || primeraPoisicon.Y == tablero.Altura
        || primeraPoisicon.X == 0 || primeraPoisicon.X == tablero.Anchura;

}
```

3.6 - Juntar las partes

Finalmente, para que todo funcione correctamente debemos juntar cada una de las partes, en un orden con sentido. Para ello dentro de la clase `main` dividiremos el código de la siguiente manera.

Primero, como hemos visto antes inicializamos las variables que necesitamos, ellas incluye, el `tablero` , la `serpiente` , el `caramelo` y la variable `haComido` .

```
Tablero tablero = new Tablero(20, 20);

Serpiente serpiente = new Serpiente(10, 10);
Caramelo caramelo = new Caramelo(0, 0);
bool haComido = false;
```

Una vez tenemos todo inicializado, debemos pasar a dibujarlo, recordamos que esta todo en un bucle `do While` . Dentro del bucle, debemos primero de todo, limpiar la pantalla, y depués dibujarlo todo. siempre y cuando la serpiente este viva.

Podemos hacer la comprobación de si esta viva tanto al principo del búcle, como al final del mismo. no importa. pero si lo hacemos al principo, debemos ejecutar los movimientos, creacion de la serpiente y el caramelo únicamente si seguimos vivos. En caso opuesto, mostraremos el mensaje de que hemos muerto y una puntuación.

Siendo el siguiente código el resultado de la clase main

```
static void Main(string[] args)
{
    Tablero tablero = new Tablero(20, 20);

    Serpiente serpiente = new Serpiente(10, 10);
    Caramelo caramelo = new Caramelo(0, 0);
    bool haComido = false;


    do
    {
        Console.Clear();
        tablero.DibujarTablero();
        //movemos y comprobamos si ha comido en el turno anterior.
        serpiente.Moverse(haComido);

        //Comprobamos si se ha comido el caramelo
        haComido = serpiente.ComeCaramelo(caramelo, tablero);

        //Dibujamos serpiente
        serpiente.DibujarSerpiente();

        //Si no contiene el caramelo, instanciamos uno nuevo.
        if (!tablero.ContieneCaramelo)
        {
            caramelo = Caramelo.CrearCaramelo(serpiente, tablero);
        }

        //Dibujamos caramelo
```

```
            caramelo.DibujarCaramelo();

            //Leemos informacion por teclado de la direccion.
            var sw = Stopwatch.StartNew();
            while (sw.ElapsedMilliseconds <= 250)
            {
                serpiente.Direccion = Util.LeerMovimiento(serpiente.Direccion);
            }

        } while (serpiente.ComprobarMorir(tablero));

        Util.MostrarPuntuación(tablero, serpiente);

        Console.ReadKey();
    }
```

Poc