

PRÁCTICA FINAL

Programación Orientada a Objetos

Adrián Belda Doñabeytia



FROGGER

*PULSA [INTRO]
PARA EMPEZAR

*PULSA [ESC]
PARA SALIR

Introducción

En esta memoria se muestra el desarrollo y resultado final del código de mi juego desarrollado en consola: Frogger. Se mostrará una breve descripción sobre la que construyo un Diagrama de Clases. Después de eso, se dividirá cada mecánica relevante y se mostrará qué hace y cómo es su código. Más adelante, la implementación de varias de estas clases combinadas. Y finalmente, adjuntaré un enlace a un repositorio donde se dispondrá de la última build del juego y su código.

Índice

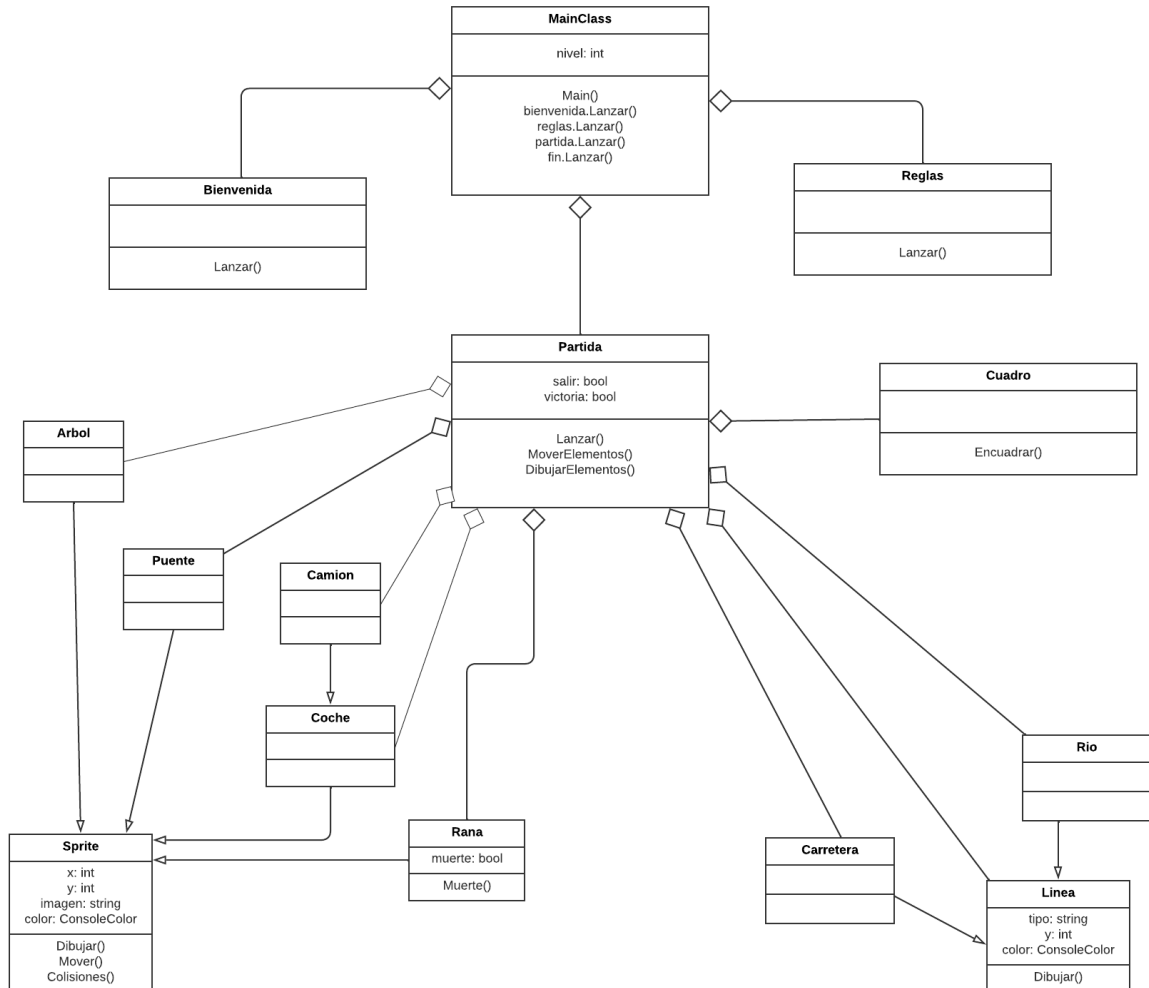
Descripción

En este juego habrá un solo jugador que puede moverse libremente por la pantalla, tanto vertical como horizontalmente, un cuadro por paso, dentro de los confines del juego. El objetivo del jugador es llegar arriba del todo para avanzar al siguiente nivel, donde deberá de repetir lo mismo, pero en un diferente mapa.

Cada nivel está dividido en diferentes terrenos con diferentes obstáculos: Los árboles pueden eliminar al jugador si se pone encima (al ser el jugador una rana, no sabe escalar árboles, así que se cae). Los coches se mueven a la derecha y pueden atropellar al jugador. El río puede hacer que el jugador se ahogue, así que tendrá que pasar por un puente.

En el juego habrá una pantalla de título, donde el jugador podrá salir del juego o empezar la partida. Antes de empezar la partida, se expondrá en una pantalla las reglas.

Diagrama de Clases



Mi idea es tener la clase main como el "Controlador del loop del juego" en vez de hacerlo lineal, por si quiero cambiar o modificar el orden u obligatoriedad de la pantalla de Reglas, por ejemplo.

La partida en sí está dividida en dos herencias: la de Sprite y la de Línea. La de línea es para el terreno, que se genera por tiras, y el de Sprite es para elementos que requieren de imagen.

El Código

Menús

Bienvenida

Antes incluso de que se haya abierto la pantalla de bienvenida, se han creado todas las pantallas en la clase de Main. Estas son la pantalla de bienvenida, las reglas, la partida y el “fin” (que es algo así como la puntuación que se muestra al final).

Manteniendo todo controlado por la misma clase madre puede ayudar en caso de querer cambiar el orden de algunas pantallas, y hace más intuitiva la programación, al servir de Hub central.

```
class MainClass
{
    0 referencias
    public static void Main()
    {
        Console.SetWindowSize(79, 24);
        Console.SetBufferSize(79, 24);
        Console.CursorVisible = false;

        Bienvenida bienvenida = new Bienvenida();
        Reglas reglas= new Reglas();
        Partida partida = new Partida();
        Fin fin = new Fin();
        int pantalla = 0;

        //Este loop sirve para que el jugador vuelva al menú principal despues de abnadar la partida o morir
        do
        {
            bienvenida.Lanzar();
            if (!bienvenida.Salir)
            {
                reglas.Lanzar();

                do
                {
                    pantalla++;
                    partida.Lanzar();

                } while (!partida.Salir);

                partida.Salir = false;
                fin.Lanzar(pantalla);
            } while (!bienvenida.Salir);
        }
    }
}
```

Fig 1: Clase MainClass

Lo primero que ve el jugador, sin embargo, es la pantalla de bienvenida. Es simple texto con caracteres ascii para el título. De la forma en que está hecha, el ESCAPE cierra el juego (al salir del loop de la clase MainClass), y cualquier otra tecla inicia el juego, aunque ponga que hay que pulsar INTRO.

```
2 referencias
public class Bienvenida
{
    3 referencias
    public bool Salir { get; set; }
    1 referencia
    public void Lanzar()
    {
        Console.SetWindowSize(79, 24);
        Console.SetBufferSize(79, 24);
        Console.Clear();
        ConsoleKey usuario;

        Console.SetCursorPosition(18, 5);
        Console.WriteLine("███");
        Console.SetCursorPosition(18, 6);
        Console.WriteLine("███");
        Console.SetCursorPosition(18, 7);
        Console.WriteLine("███");
        Console.SetCursorPosition(18, 8);
        Console.WriteLine("███");
        Console.SetCursorPosition(18, 9);
        Console.WriteLine("███");

        Console.SetCursorPosition(20, 12);

        Console.WriteLine("      *PULSA [INTRO]");
        Console.SetCursorPosition(20, 13);
        Console.WriteLine("      PARA EMPEZAR");

        Console.SetCursorPosition(20, 19);
        Console.WriteLine("      *PULSA [ESC]");
        Console.SetCursorPosition(20, 20);
        Console.WriteLine("      PARA SALIR");

        //Si el usuario le da a Escape, se activa la variable salir (que toma efecto en la clase Juego. Si pulsa R, se va al Bienvenida.Reglas)
        usuario = Console.ReadKey(true).Key;

        if (usuario == ConsoleKey.Escape)
        {
            Salir = true;
        }
    }
}
```

Fig 2: Clase Bienvenida

El jugador lo ve así:



Fig 3: Bienvenida

Reglas

Una vez el jugador le da a cualquier tecla menos ESC, la pantalla cambia a la de Reglas, donde se explica qué hace cada cosa y el objetivo del jugador. La única forma de progresar en este menú es darle a la tecla INTRO (esta vez no vale cualquier tecla, para asegurarme de que el jugador no se ha saltado sin querer las reglas).

```
//Ubicaria hacia esta ronda ya llega el jump
public class Reglas
{
    public void Lanzar()
    {
        //Ubicaria hacia esta ronda ya llega el jump
        Console.Clear();
        Console.SetCursorPosition(0, 30);
        Console.SetCursorPosition(0, 30);

        Console.SetCursorPosition(10, 1);
        Console.WriteLine("¡¡ G U I A !!");
        Console.SetCursorPosition(5, 4);
        Console.WriteLine("■ = 10");

        Console.SetCursorPosition(1, 5);
        Console.WriteLine("-----");

        Console.SetCursorPosition(13, 6);
        Console.WriteLine("Cosas que te matan:");

        Console.BackgroundColor = ConsoleColor.Gray;
        Console.SetCursorPosition(2, 7);
        Console.WriteLine(" ");

        Console.SetCursorPosition(2, 8);
        Console.BackgroundColor = ConsoleColor.Black;
        Console.WriteLine("■");
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("■");
        Console.BackgroundColor = ConsoleColor.Black;
        Console.WriteLine("■");
        Console.ForegroundColor = ConsoleColor.White;
        Console.BackgroundColor = ConsoleColor.Black;
        Console.WriteLine("■ = Coche");

        Console.BackgroundColor = ConsoleColor.Gray;
        Console.SetCursorPosition(2, 9);
        Console.WriteLine(" ");

        Console.SetCursorPosition(2, 10);
        Console.BackgroundColor = ConsoleColor.Black;
        Console.WriteLine("■");
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.WriteLine("■");
        Console.BackgroundColor = ConsoleColor.Black;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("■ = Camión");

        Console.BackgroundColor = ConsoleColor.Gray;
        Console.SetCursorPosition(2, 11);
        Console.WriteLine(" ");

        Console.SetCursorPosition(5, 12);
        Console.BackgroundColor = ConsoleColor.Green;
        Console.ForegroundColor = ConsoleColor.DarkGreen;
        Console.WriteLine("■");
        Console.BackgroundColor = ConsoleColor.Black;
        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine("■ = Árbol (sí, te mata también, por tonto)");

        Console.BackgroundColor = ConsoleColor.Gray;
        Console.SetCursorPosition(2, 13);
        Console.WriteLine(" ");

        Console.SetCursorPosition(5, 14);
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.WriteLine("■");
        Console.BackgroundColor = ConsoleColor.Black;
        Console.WriteLine("■ = Río");

        Console.SetCursorPosition(2, 17);
        Console.WriteLine("-----");

        Console.SetCursorPosition(18, 18);
        Console.WriteLine("CONTROLES");

        Console.SetCursorPosition(5, 20);
        Console.WriteLine("WASD - Movimiento");
        Console.WriteLine("Escape - Salir");

        Console.SetCursorPosition(18, 23);
        Console.WriteLine("PULSA INTRO PARA EMPEZAR");
        Console.ReadKey();
    }
}
```

Fig 4: Clase Reglas

```

R E G L A S

■ = TÚ
-----
Cosas que te matan:
■ = Coche
■ = Camión
■ = Árbol (sí, te mata también, por tonto)
■ = Río
-----
CONTROLES

WASD - Movimiento
Escape - Salir

PULSA INTRO PARA EMPEZAR
```

Fig 5: Reglas

Antes de proceder a la clase partida, conviene explicar la generación de terreno y de sprites, comenzando con la de terreno:

Líneas y Obstáculos

Las Líneas son las tiras de terreno generadas a lo largo del mapa. Hay tres tipos: Línea (la básica, representa una pradera con árboles), Carretera y Río. Ahora explicaremos los elementos que tienen

Línea Básica

La Línea básica de la que heredan el resto de líneas es una tira de 35 de largo (toda lo largo posible) y 2 de alto, como el resto de líneas. Ésta tiene color verde y es de tipo "línea".

Éste es su código de generación:

```
9 referencias
class Línea
{
    protected string tipo;
    protected int y;
    protected ConsoleColor color;

    0 referencias
    public int PosY
    {
        get { return y; }
        set { y = value; }
    }

    0 referencias
    public Línea()
    {
        tipo = "línea";
        y = 1;
        color = ConsoleColor.Green;
    }

    3 referencias
    public Línea(int lineanumero)
    {
        tipo = "línea";
        y = 1 + (2 * lineanumero);
        color = ConsoleColor.Green;
    }

    2 referencias
    public virtual void Dibujar()
    {
        Console.BackgroundColor = color;

        for (int i = 0; i < 2; i++)
        {
            Console.SetCursorPosition(4, y + i);
            for (int j = 0; j < 35; j++)
            {
                Console.Write(" ");
            }
        }

        Console.BackgroundColor = ConsoleColor.Black;
    }
}
```

Fig 6: Clase Línea

Carretera

Línea de color gris de tipo "carretera".


```

3 referencias
class Carretera : Linea
{
    0 referencias
    public Carretera()
    {
        tipo = "carretera";
        y = 1;
        color = ConsoleColor.Gray;
    }
    1 referencia
    public Carretera(int lineanumero)
    {
        tipo = "carretera";

        y = 1 + (2 * lineanumero);
        color = ConsoleColor.DarkGray;
    }
}

```

Fig 7: Clase Carretera

Rio

Línea de color azul de tipo "rio". Esta línea de por sí es letal para el jugador, pero eso se implementará en la clase de Partida (vea el índice).

```

3 referencias
class Rio : Linea
{
    0 referencias
    public Rio()
    {
        tipo = "rio";
        y = 1;
        color = ConsoleColor.Blue;
    }
    1 referencia
    public Rio(int lineanumero)
    {
        tipo = "rio";

        y = 1 + (2 * lineanumero);
        color = ConsoleColor.Blue;
    }
}

```

Fig 8: Clase Rio

Sprites

Los sprites son todos los elementos interactivables del juego. Todos ellos comparten una clase abstracta homónima:

Sprite

Esta clase abstracta tiene las funciones básicas que casi todos los elementos tienen: Dibujar (permite renderizarlos), Movimiento en todas las direcciones y la detección de Colisiones basada en longitud.

Al ser todos los elementos de este juego (salvo el puente) unidimensionales (que se extienden solo en el vector x), pues con la longitud solamente se puede calcular el rango de colisiones.

Aquí está el código:

```
5 references
abstract class Sprite
{
    protected int x, y;
    protected string imagen;
    protected ConsoleColor color;

    2 references
    public int PosY { get { return y; } set { y = value; } }
    3 references
    public int PosX { get { return x; } set { x = value; } }
    6 references
    public void MoverA(int nuevaX, int nuevaY)
    {
        x = nuevaX;
        y = nuevaY;
    }

    8 references
    public virtual void Dibujar()
    {
        Console.ForegroundColor = color;
        Console.SetCursorPosition(x, y);
        Console.Write(imagen);
        Console.CursorVisible = false;
    }

    3 references
    public virtual void MoverDerecha()
    {
        x++;
        if (x >= 38) x = 38;
    }

    2 references
    public virtual void MoverIzquierda()
    {
        x--;
        if (x <= 0) x = 0;
    }

    1 reference
    public void MoverArriba()
    {
        y--;
        if (y <= 1) y = 1;
    }

    1 reference
    public void MoverAbajo()
    {
        y++;
        if (y >= 22) y = 22;
    }

    7 references
    protected abstract int DevolverLongitud();
    2 references
    public bool ColisionaCon(Sprite sprite)
    {
        if (this.y != sprite.y)
        {
            return false;
        }
        else
        {
            for (int i = 0; i < this.DevolverLongitud(); i++)
            {
                for (int j = 0; j < sprite.DevolverLongitud(); j++)
                {
                    if ((this.x + i) == (sprite.x + j))
                        return true;
                }
            }
        }
        return false;
    }
}
```

Fig 9: Clase Sprite

Rana

Éste es el sprite que encarna el jugador. Tiene la exclusiva clase “Death()”, que controla la “animación” de muerte, que señala exactamente donde murió el jugador, para evitar confusiones y la desorientación del jugador:

```
class Rana : Sprite
{
    bool muerte = false;

    1 referencia
    public bool Muerte
    {
        get { return muerte; }
        set { muerte = value; }
    }

    1 referencia
    public Rana()
    {
        imagen = "■";
        x = 20;
        y = 20;
        color = ConsoleColor.White;
    }

    3 referencias
    protected override int DevolverLongitud()
    {
        return 1;
    }

    3 referencias
    public void Death()
    {
        imagen = "X";
        color = ConsoleColor.Red;
        muerte = true;
    }
}
```

Fig 10: Clase Rana

Coche

Los coches son obstáculos encontrados en las Carreteras. Estos obstáculos se mueven a la derecha dejando unos huecos por donde el jugador tiene que pasar sin ser tocado por uno de estos.

La clase coche es más larga que el resto, así que la voy a dividir en varios trozos.

```
class Coche : Sprite
{
public:
    Coche()
    {
        Random random = new Random();
        int rand = random.Next(0, 3);
        if (rand == 0)
        {
            color = ConsoleColor.Blue;
        }
        if (rand == 1)
        {
            color = ConsoleColor.Green;
        }
        if (rand == 2)
        {
            color = ConsoleColor.Red;
        }
        if (rand == 3)
        {
            color = ConsoleColor.Yellow;
        }
    }

    protected override int DrawLength()
    {
        return 1;
    }

    public Coche(int x, int y)
    {
        x = x;
        y = y;
        Random random = new Random();
        int rand = random.Next(0, 3);
        if (rand == 0)
        {
            color = ConsoleColor.Blue;
        }
        if (rand == 1)
        {
            color = ConsoleColor.Green;
        }
        if (rand == 2)
        {
            color = ConsoleColor.Red;
        }
        if (rand == 3)
        {
            color = ConsoleColor.Yellow;
        }
    }

    public override void Dibujar()
    {
        Console.SetCursorPosition(x, y);
        Console.ForegroundColor = ConsoleColor.Black;
        //con esto se evita que los coches se dibujen fuera o por encima del canvas
        if (x < 0 || x > 20)
        {
            Console.WriteLine("");
        }
        if (x > 0 && x < 20)
        {
            Console.ForegroundColor = color;
            Console.WriteLine(" ");
        }
        if (x > 20 && x < 30)
        {
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine(" ");
        }
    }

    public override void MoverDerecha()
    {
        x++;
        if (x > 30) x = 1;
    }

    public override void MoverIzquierda()
    {
        x--;
        if (x < 0) x = 20;
    }
}
```

Fig 11: Clase Coches

La primera parte del código consiste en dos constructores: uno por defecto para debugging y otro, el que se utiliza, que coloca el coche en una posición de x y y. Estos constructores se encargan de dar un color aleatorio a los coches mediante una variable random:

```

0 referencias
public Coche()
{
    Random random = new Random();
    int rand = random.Next(0, 4);
    if (rand == 0)
    {
        color = ConsoleColor.Blue;
    }
    if (rand == 1)
    {
        color = ConsoleColor.Green;
    }
    if (rand == 2)
    {
        color = ConsoleColor.Red;
    }
    if (rand == 3)
    {
        color = ConsoleColor.Yellow;
    }
}

4 referencias
protected override int DevolverLongitud()
{
    return 3;
}

1 referencia
public Coche(int h, int v)
{
    x = h;
    y = v;
    Random random = new Random();
    int rand = random.Next(0, 4);
    if (rand == 0)
    {
        color = ConsoleColor.Blue;
    }
    if (rand == 1)
    {
        color = ConsoleColor.Green;
    }
    if (rand == 2)
    {
        color = ConsoleColor.Red;
    }
    if (rand == 3)
    {
        color = ConsoleColor.Yellow;
    }
}
4 referencias

```

Fig. 12: Clase Coches, Random

Después, están las funciones estándar de Dibujar y Mover, que hacen overwrite a la clase Sprite para ajustarse.

La función de Dibujar tiene unos parámetros que impiden que el coche se renderice una vez ha salido del espacio visible de la pantalla, y la función de MoverDerecha teletransporta los coches a la izquierda de la pantalla al salir completamente de la visión del jugador.

```

4 referencias
public override void Dibujar()
{
    Console.SetCursorPosition(x, y);

    Console.ForegroundColor = ConsoleColor.Black;
    //CON ESTOS IFS, NOS ASEGURAMOS QUE LOS COCHES NO SE DIBUJEN FUERA O POR ENCIMA DEL CUADRO
    if (x > 5 && x < 39)
    {
        Console.Write("■");
    }
    if (x + 1 > 5 && x + 1 < 39)
    {
        Console.ForegroundColor = color;
        Console.Write("■");
    }
    if (x + 2 > 5 && x + 2 < 39)
    {
        Console.ForegroundColor = ConsoleColor.Black;
        Console.Write("■");
    }
}

3 referencias
public override void MoverDerecha()
{
    x++;
    if (x >= 38) x = 1;
}

```

Fig. 13: Clase Coches, Dibujar y Mover

Camión

La clase Camión es exactamente igual a la del Coche, salvo que renderiza algo parecido a un camión y es un solo bloque más grande que el coche. Ésta hereda de la clase Coche.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApp1
{
    class Camion : Coche
    {
        //Constructor
        public Camion()
        {
            Random random = new Random();
            int rand = random.Next(0, 4);
            if (rand == 0)
            {
                color = ConsoleColor.Blue;
            }
            if (rand == 1)
            {
                color = ConsoleColor.Green;
            }
            if (rand == 2)
            {
                color = ConsoleColor.Red;
            }
            if (rand == 3)
            {
                color = ConsoleColor.Yellow;
            }
        }

        //Constructor
        public Camion(int h, int w)
        {
            x = h;
            y = w;
            Random random = new Random();
            int rand = random.Next(0, 4);
            if (rand == 0)
            {
                color = ConsoleColor.Blue;
            }
            if (rand == 1)
            {
                color = ConsoleColor.Green;
            }
            if (rand == 2)
            {
                color = ConsoleColor.Red;
            }
            if (rand == 3)
            {
                color = ConsoleColor.Yellow;
            }
        }

        //Método
        public override void Dibujar()
        {
            Console.SetCursorPosition(x, y);
            Console.ForegroundColor = ConsoleColor.Black;

            if (x > 0 && x < 30)
            {
                Console.Write("┌");
            }
            if (x+1 > 0 && x+1 < 30)
            {
                Console.Write("┐");
            }

            if (x+2 > 0 && x+2 < 30)
            {
                Console.Write("┌");
            }
            if (x+3 > 0 && x+3 < 30)
            {
                Console.ForegroundColor = color;
                Console.Write("┐");
            }
        }

        //Método
        protected override int DevolverLongitud()
        {
            return 4;
        }
    }
}
```

Fig. 14: Clase Camión

Árbol

La clase Árbol genera unos obstáculos estáticos en las Líneas verdes. Éstos matan al jugador si los toca.

```
5 referencias
class Arbol : Sprite
{
    0 referencias
    public Arbol()
    {
        imagen = "🌳";
        color = ConsoleColor.DarkGreen;
    }

    1 referencia
    public Arbol(int h,int v)
    {
        x = h;
        y = v;
        imagen = "🌳";
        color = ConsoleColor.DarkGreen;
    }

    3 referencias
    protected override int DevolverLongitud()
    {
        return 1;
    }

    3 referencias
    public override void Dibujar()
    {
        Console.SetCursorPosition(x, y);
        Console.BackgroundColor = ConsoleColor.Green;
        Console.ForegroundColor = color;
        Console.WriteLine(imagen);
    }
}
```

Fig. 15: Clase Árbol

Tronco

El tronco es un sprite cuya única función es conformar una parte del puente, para cruzar el río. Su código es muy básico.

```
6 referencias
class Tronco : Sprite
{
    0 referencias
    public Tronco()
    {
        imagen = "🌳";
        color = ConsoleColor.DarkRed;
    }

    2 referencias
    public Tronco(int h, int v)
    {
        x = h;
        y = v;
        imagen = "🌳";
        color = ConsoleColor.DarkRed;
    }

    3 referencias
    protected override int DevolverLongitud()
    {
        return 1;
    }

    3 referencias
    public override void Dibujar()
    {
        Console.SetCursorPosition(x, y);
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.ForegroundColor = color;
        Console.WriteLine(imagen);
    }
}
```

Fig. 16: Clase Tronco

Bloques

Los grupos se crean con intención de optimizar código y hacer los sprites más manejables.

Bloque de Coches

El bloque de coches contiene a toda una fila horizontal de coches, y controla su movimiento. Al igual que el código de los coches, éste es extenso a fin de dar variedad a las filas de coches, así que lo partiremos en dos partes.

Éste es el código entero:

```
class BloqueCoches
{
    int x = 1, i = 0;
    Random random = new Random();

    //Constructor
    public BloqueCoches(int w)
    {
        Coches = new Coche[6];

        /*Cursor en x=0, crear coche o camion, avanzar 3/4/5, crear otro hasta llegar al ultimo visible w/
        do
        {
            Console.SetCursorPosition(x, w);
            int rand = random.Next(0, 4);
            if (rand == 0)
            {
                Coches[i] = new Coche(x, w);
                x = x + 3;
            }
            else
            {
                Coches[i] = new Camion(x, w);
                x = x + 4;
            }
            i++;
            rand = random.Next(0, 3);
            if (rand == 0)
            {
                x = x + 3;
            }
            if (rand == 1)
            {
                x = x + 4;
            }
            if (rand == 2)
            {
                x = x + 5;
            }
        } while (x < 39);

        //Propiedades
        public Coche[] Coches { get; set; }
        //Metodos
        public void Dibujar()
        {
            for (int j = 0; j < i; j++)
            {
                Coches[j].Dibujar();
            }
        }
        public void MoverDerecha()
        {
            for (int j = 0; j < i; j++)
            {
                Coches[j].MoverDerecha();
            }
        }
    }
}
```

Fig. 17: Clase BloqueCoches

La primera parte del código muestra un random. Su propósito es decidir si se va a generar un coche o un camión. Seguido de esto, se vuelve a utilizar otro número generado para saber cuántos espacios tendrá de separación el coche que el siguiente. Con estos dos parámetros, se consigue que los coches no parezcan demasiado ordenados.

Se crearán un total de 6 coches a menos que el cursor con el que se crean los coches llegue a la derecha del todo, momento en que dejará de generarlos.


```

1 referencia
public BloqueCoches(int v)
{
    Coches = new Coche[6];

    /*Cursor en x=i, crear coche o camion, avanzar 3/4/5, crear otro hasta llegar al ultimo visible */
    do
    {
        Console.SetCursorPosition(x, v);
        int rand = random.Next(0, 4);
        if (rand <= 2)
        {
            Coches[i] = new Coche(x, v);
            x = x + 3;
        }
        else
        {
            Coches[i] = new Camion(x, v);
            x = x + 4;
        }
        i++;
        rand = random.Next(0, 3);
        if (rand == 0)
        {
            x = x + 3;
        }
        if (rand == 1)
        {
            x = x + 4;
        }
        if (rand == 2)
        {
            x = x + 5;
        }
    } while (x < 39);
}

7 referencias
public Coche[] Coches { get; set; }
1 referencia

```

Fig. 18: Clase BloqueCoches, Generación de Coches

La segunda parte del código son las funciones de Dibujar y de MoverDerecha de los coches, pero este recorre todo el array invocando estos métodos a los coches individualmente:

```

1 referencia
public void Dibujar()
{
    for (int j = 0; j < i; j++)
    {
        Coches[j].Dibujar();
    }
}

1 referencia
public void MoverDerecha()
{
    for (int j = 0; j < i; j++)
    {
        Coches[j].MoverDerecha();
    }
}

```

Fig. 19: Clase BloqueCoches, Dibujar y Mover

Así es como se ve el conjunto de la Carretera, los Coches, Camiones y los BloqueCoches implementados:



Bloque de Árboles

El grupo de BloqueArboles genera aleatoriamente 10 árboles por la Línea verde.

```
class BloqueArboles
{
    int k = 0;
    Random random = new Random();

    1 referencia
    public BloqueArboles(int v)
    {
        k = 0;
        Arboles = new Arbol[10];
        for (int j = 0; j < 2; j++)
        {
            for (int i = 0; i < 5; i++)
            {
                int rand = random.Next(4, 38);
                Console.SetCursorPosition(rand, v+j);
                Arboles[k] = new Arbol(rand, v+j);
                k++;
            }
        }
    }

    1 referencia
    public void Dibujar()
    {
        for (int i = 0; i < k; i++)
        {
            Arboles[i].Dibujar();
        }
    }

    5 referencias
    public Arbol[] Arboles { get; set; }
}
6 referencias
```

Fig. 20: Clase BloqueArboles

Así es como se ve el conjunto de la Línea, los Árboles y los BloqueÁrboles implementados:



Puente

A pesar que este no tiene en su nombre “Bloque”, el puente es un bloque de troncos. Éste se encarga de colocar los troncos en un 3x2 modular (así que puede ser más grande si dos ríos están al lado) en medio de todo el mapa. Andar por este puente impedirá al jugador morir por el agua.

```
class Puente
{
    int k = 0;

    1 referencia
    public Puente(int v)
    {
        k = 0;
        Troncos = new Tronco[12];
        for (int j = 0; j < 3; j++)
        {
            for (int i = 0; i < 2; i++)
            {
                Console.SetCursorPosition(19+j, v);
                Troncos[k] = new Tronco(19+j, v);
                k++;
                Console.SetCursorPosition(19+j, v + 1);
                Troncos[k] = new Tronco(19+j, v + 1);
                k++;
            }
        }
    }

    1 referencia
    public void Dibujar()
    {
        for (int i = 0; i < k; i++)
        {
            Troncos[i].Dibujar();
        }
    }

    4 referencias
    public Tronco[] Troncos { get; set; }
}
```

Fig. 21: Clase Puente

Así es como se ve el conjunto del Río, los Troncos y el Puente implementados:



El Encuadre

La clase Cuadro se encarga de hacer que los bordes del espacio visible para el jugador sean más estéticamente agradables. Simplemente dibuja unas líneas a los bordes.

```
public class Cuadro
{
    1 referencia
    public void Encuadrar()
    {
        //EL CUADRO MIDE 38 DE LARGO CON MARGEN DE 2 Y 24 DE ALTO, SIN MARGEN
        //PARTE DE ARRIBA
        Console.SetCursorPosition(3, 0);
        Console.BackgroundColor = ConsoleColor.Black;
        Console.ForegroundColor = ConsoleColor.White;
        Console.Write("┌");

        for (int i = 0; i < 35; i++)
        {
            Console.Write("═");
        }
        Console.Write("┐");

        //PARTE DEL MEDIO
        for (int i = 0; i < 22; i++)
        {
            Console.SetCursorPosition(3, i + 1);
            Console.Write("│");
            Console.SetCursorPosition(39, i + 1);
            Console.Write("│");
        }

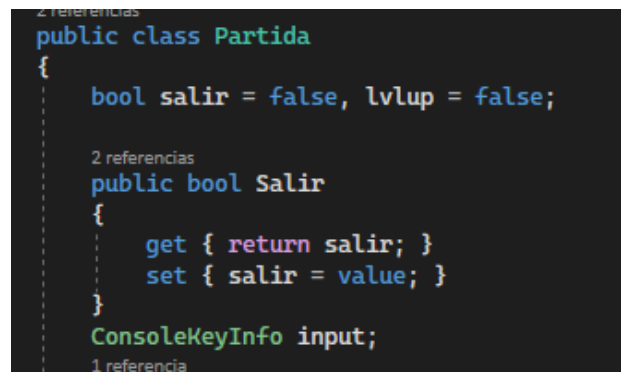
        //PARTE DE ABAJO
        Console.SetCursorPosition(3, 23);
        Console.Write("└");
        for (int i = 0; i < 35; i++)
        {
            Console.Write("═");
        }
        Console.Write("┘");
    }
}
```

Fig. 22: Clase Cuadro

La Partida

Sin duda, la clase más extensa de todas. Es la clase que controla la partida y que une todos los módulos antes mencionados en uno sólo. Debido a la naturaleza aleatoria del juego, casi todos los procesos de generación arbitraria se hacen aquí. Procederé a explicar el código por partes.

Primero, se crean todas las variables. En este caso, hay dos importantes que ambas son booleanos: “salir” y “lvlup”. Ambos de esos booleanos veremos que causan que se salga del loop del juego (el que representa cada frame). La diferencia es que el “lvlup” lo que hace es que se vuelva a cargar el mapa, para un nuevo nivel aleatorio, mientras que el “salir” es comunicado a la clase MainClass para indicar que muestre la pantalla “Fin” y vuelva al menú de bienvenida.

A screenshot of a code editor showing the C# code for the Partida class. The code is as follows:

```
2 referencias
public class Partida
{
    bool salir = false, lvlup = false;

    2 referencias
    public bool Salir
    {
        get { return salir; }
        set { salir = value; }
    }

    ConsoleKeyInfo input;
    1 referencia
```

Fig. 23: Clase Partida, Variables

Después tenemos la creación de los sprites, las líneas y su asignación de tipo. Lo primero que se generan son las líneas.

En este caso se generan 11 en total. La primera y la última, por motivos de impedir que el jugador aparezca en un obstáculo y sea instantáneamente eliminado, son Líneas Básicas sin ningún obstáculo. El resto de líneas se generan mediante un random. Cada vez que se crea una línea, también se crea su correspondiente Grupo de obstáculos.

```

Cuadro cuadro = new Cuadro();
Linea[] linea = new Linea[11];
BloqueCoches[] bloqueCoches = new BloqueCoches[8];
BloqueArboles[] bloqueArboles = new BloqueArboles[8];
Punte[] puente = new Punte[8];

//Es necesario saber el valor Y de los ríos para calcular su colisión con el jugador
int[] posRios = new int[16];

//Se crean valores independientes para cada Grupo de Sprites para tener una correcta notación de sus correspondientes arrays.
int rand, numcoches = 0, numarboles = 0, numtroncos = 0;
Random rng = new Random();

//Este for crea 11 líneas en total aleatorias, dejando la primera y la última como básicas verdes
for (int i = 0; i < linea.Length; i++)
{
    if (i >= 10)
    {
        linea[i] = new Linea(i);
    }
    else if (i == 0)
    {
        linea[i] = new Linea(i);
    }
    else
    {
        rand = (rng.Next(0, 3));
        if (rand == 0)
        {
            linea[i] = new Linea(i);
            bloqueArboles[numarboles] = new BloqueArboles(1 + (2 * i));
            numarboles++;
        }
        if (rand == 1)
        {
            linea[i] = new Carretera(i);
            bloqueCoches[numcoches] = new BloqueCoches(1 + (2 * i));
            numcoches++;
        }
        if (rand == 2)
        {
            linea[i] = new Rio(i);
            puente[numtroncos] = new Punte(1 + (2 * i));
            posRios[numtroncos] = 1 + (2 * i);
            numtroncos++;
        }
    }
}

//Aquí recorro todo el array de las posiciones del río sirviendome de la variable numtroncos, que es igual que el largo del array para adivinar cuales son
//todas las posiciones Y que son río.
for (int i = numtroncos+1; i <= numtroncos+2; i++)
{
    posRios[i] = posRios[(i - numtroncos) - 1] + 1;
}

```

Fig. 24: Clase Partida, Generación de Líneas y Obstáculos

Posteriormente, se genera todo por primera vez antes de entrar al loop del juego:

```

Rana rana = new Rana();

Console.Clear();
//se genera el Encuadre solamente una vez
cuadro.Encuadrar();

for (int i = 0; i < linea.Length; i++)
{
    linea[i].Dibujar();
}

```

Lo primero del loop es el renderizado y movimiento de todos los elementos creados. Además, se aprovecha los For para calcular la colisión de cada elemento.

```

//-----LOOP-----
do
{
    //línea[v].Dibujar();
    for (int i = 0; i < línea.Length; i++)
    {
        línea[i].Dibujar();
    }
    for (int i = 0; i < numarboles; i++)
    {
        bloqueArboles[i].Dibujar();

        //Colisión de la Rana con los Árboles
        for (int j = 0; j < 6; j++)
        {
            if (bloqueArboles[i] != null && bloqueArboles[i].Arboles[j] != null && bloqueArboles[i].Arboles[j].ColisionaCon(rana))
            {
                rana.Death();
            }
        }
    }
    for (int i = 0; i < numtroncos; i++)
    {
        puente[i].Dibujar();
    }
    for (int i = 0; i < numcoches; i++)
    {
        bloqueCoches[i].MoverDerecha();
        bloqueCoches[i].Dibujar();

        //Colisión de la Rana con los Coches
        for (int j = 0; j < 6; j++)
        {
            if (bloqueCoches[i].Coches[j] != null && bloqueCoches[i].Coches[j].ColisionaCon(rana))
            {
                rana.Death();
            }
        }
    }
}

//Aquí establezco que si la rana se encuentra en una posición registrada en el Array de posiciones del río, es que está en el río.
//Además, si la rana está en X=19, 20 y 21 significaría que está en el puente, así que no triggera el Game Over
if (Array.Exists(posRios, element => element == rana.PosY && rana.PosX != 19 && rana.PosX != 20 && rana.PosX != 21)
{
    rana.Death();
}
}

```

Fig. 25: Clase Partida, Renderizado, Movimiento y Colisión de los Sprites

A continuación, tenemos la última parte del loop, que muestra la condición para que el jugador pueda pasar de nivel (llegar arriba del todo).

Seguido de esto, se renderiza la rana, se oculta el cursor y se registra la tecla que presiona el jugador. En caso de que presione ESCAPE, le mostrará el menú de “Fin” y le llevará a la pantalla de bienvenida. En caso de que muera, también pasará lo mismo, pero se mostrará una **X** roja donde murió.


```

//CONDICIÓN PARA PASAR DE NIVEL
if (rana.PosY <= 1)
{
    lvlup = true;
}

Console.BackgroundColor = ConsoleColor.Black;
rana.Dibujar();

//Esto es para que no aparezca la letra pulsada
Console.SetCursorPosition(0, 0);
Console.ForegroundColor = ConsoleColor.Black;

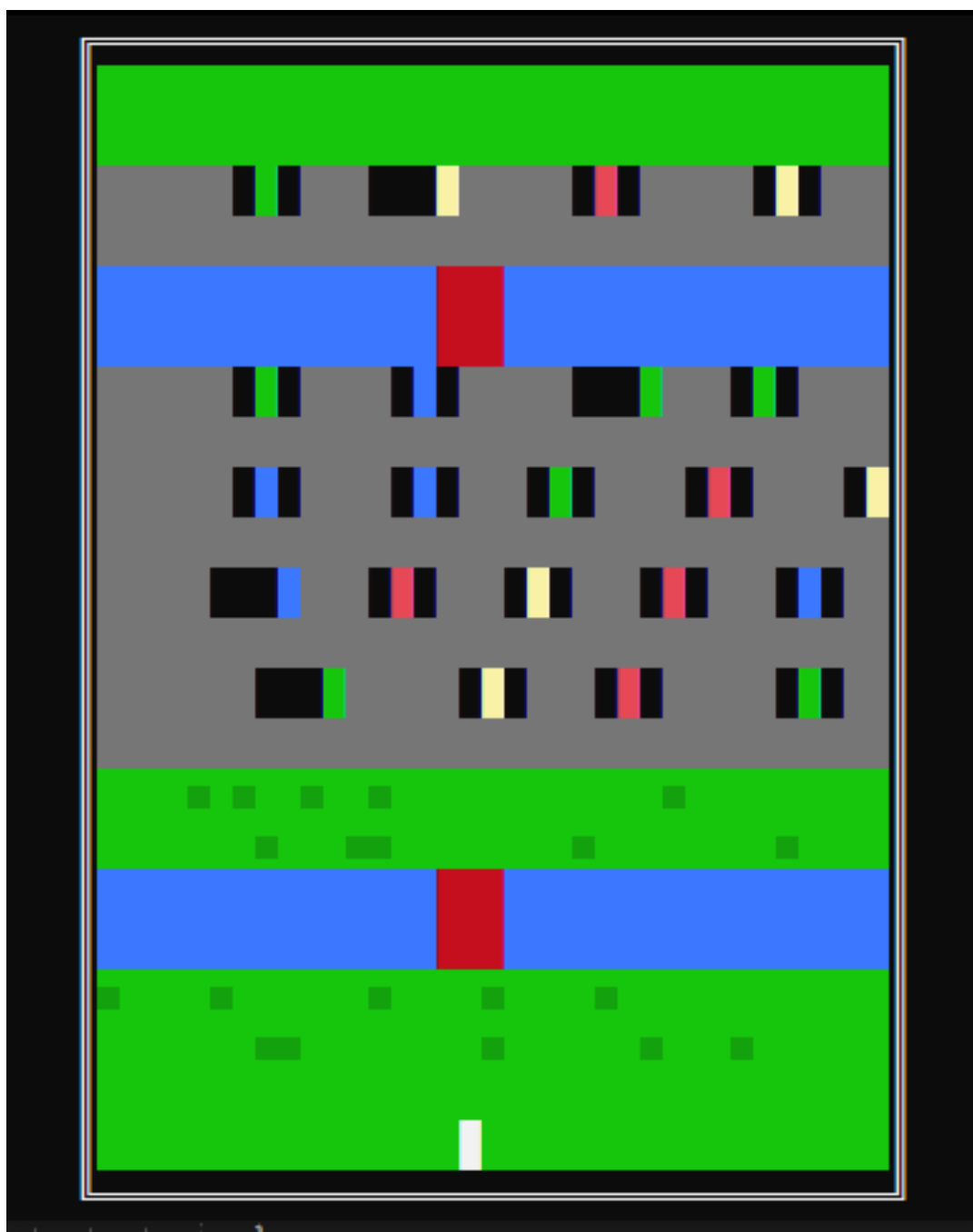
if (Console.KeyAvailable)
{
    input = Console.ReadKey();
    if (input.Key == ConsoleKey.W)
    {
        rana.MoverArriba();
    }
    if (input.Key == ConsoleKey.A)
    {
        rana.MoverIzquierda();
    }
    if (input.Key == ConsoleKey.S)
    {
        rana.MoverAbajo();
    }
    if (input.Key == ConsoleKey.D)
    {
        rana.MoverDerecha();
    }
    if (input.Key == ConsoleKey.Escape)
    {
        salir = true;
    }
}
Thread.Sleep(50);

//Si en algún momento la rana se murió, cuando ya se haya renderizado todo el jugador puede ver dónde murió antes de salir del juego.
if (rana.Muerte)
{
    Console.ReadLine();
    salir = true;
}
} while (salir == false && lvlup == false);
}

```

Fig. 26: Clase Partida, Final del Loop

Toda esta clase produce una imagen tal que así:



Pantalla de Fin de Partida

La pantalla de Final de Partida es la más simple, pues solamente muestra hasta qué nivel ha llegado el jugador:

```
2 referencias
public class Fin
{
    int nivel = 0;

    1 referencia
    public void Lanzar(int Lvl)
    {
        nivel = Lvl;
        Console.Clear();
        Console.BackgroundColor = ConsoleColor.Black;
        Console.ForegroundColor = ConsoleColor.White;
        Console.SetCursorPosition(0, 0);

        Console.WriteLine("Has llegado al nivel {0}!", nivel);
        Console.ReadLine();
    }
}
```

Fig. 27: Clase Fin

Y se ve tal que así:

¡Has llegado al nivel 2!