Universidad de Huelva

Universidad de Huelva

Grado en Ingeniería Informática

Programación de Juegos 2D en Unity

Memoria

Autor: Alberto Fernández Merchán
Asignatura: Programación de Juegos
Profesor: Javier Martín Moreno

Índice

1. Introducción **3**

2. Sistema de guardado de Datos **3**

3. Fondo del Juego **5**
3.1. Tiling 5
3.2. Parallaxing 6

4. Implementación de Interfaces **7**
4.1. Menú de Pausa 7
4.2. Indicador de vida 7

5. Inteligencia Artificial **8**
5.1. Búsqueda de Caminos 8
5.1.1. A* 8
5.1.2. Breadcrumb Pathfinding 8

6. Sugerencias Implementadas **9**

7. Recursos **9**

Índice de figuras

1. Imagen del videojuego sin utilizar la técnica de tiling 5
2. Imagen del videojuego utilizando la técnica de tiling 6
3. Orden de las capas utilizando parallaxing 6
4. Menú de pausa del juego. 7
5. Indicadores de salud en el juego 7
6. Técnicas de Búsqueda de Caminos 8

1. Introducción

Unity es una de las herramientas más utilizadas por los desarrolladores de videojuegos debido a su facilidad de uso y la capacidad de crear juegos para múltiples plataformas. Es por eso que, en esta primera práctica, utilizaré este motor de videojuegos para crear uno en dos dimensiones del género de plataformas llamado *Gravity Shift*.

Gravity Shift será un videojuego de plataformas simple en el que el personaje tendrá que invertir la gravedad para poder avanzar en el nivel. Además, también habrá zonas donde recibirá daño al pisar y un par de enemigos (terrestre y volador) que lo perseguirán y dispararán causándole también un punto de daño.
Finalmente, para completar el nivel, el jugador deberá llegar al portal situado al final del mismo con, al menos, un punto de vida.

En esta memoria, se describirán las implementaciones más interesantes a la hora de programar la lógica del juego.

2. Sistema de guardado de Datos

A la hora de guardar los datos de configuración, se ha utilizado la clase **PlayerPrefs** para poder cargarlos cuando el usuario le de al botón de *Cargar Partida* en el menú principal.
Para cargar los datos de *PlayerPrefs* se utiliza la siguiente función:

```
public void LoadGameDialogYes()
{
    if (PlayerPrefs.HasKey("SavedLevel"))
    {
        levelToLoad = PlayerPrefs.GetString("SavedLevel");
        Player_Control.puntos = PlayerPrefs.GetInt("Puntos");
        Player_Control.vida = PlayerPrefs.GetInt("Vida");
        SceneManager.LoadScene(levelToLoad);
    }
    else
        noSavedGameDialog.SetActive(true);
}
```

Además, cuando el jugador completa el juego también se guarda sus puntos si han sido mayores que los 10 mejores records conseguidos. Para ello se ha creado la clase **Record** que registra los puntos y el nombre de cada jugador. También se ha sobreescrito el método para comparar dos objetos de la misma clase (*CompareTo*). La clase Record es la siguiente:

```
public class Record : IComparable
{
    private readonly int puntos;
    private readonly string nombre;
    public Record(int puntos, string nombre)
    {
        this.puntos = puntos;
        this.nombre = nombre;
    }

    public int getPuntos()
    {
        return puntos;
    }

    public override string ToString()
    {
        return this.nombre + "\t\t\t\t\t\t\t\t\t\t\t\t" + this.puntos;
    }

    public int CompareTo(object x)
    {
        if (x == null) return 1;
        Record otherRecord = x as Record;
        if(otherRecord != null)
            return this.puntos.CompareTo(otherRecord.puntos);
        else
            throw new ArgumentException("Object is not a Record!");
    }
}
```

Para añadir el nombre al fichero de puntuación se utiliza el método *SaveName()* en la clase *ScoreControl*:

```
    public void SaveName()
    {
        string player_name = input.text;
        int numero_lineas = HandleText.NumberLines("score");
        List<Record> records = new List<Record>();
        // Lee los records
        for(int l = 0; l < numero_lineas; l++)
        {
            string line = HandleText.ReadLine(l, "score").Replace("\t\t\t\t\t\t\t\t\t\t\t\t", " ");
            string[] line_split = line.Split(" ");
            records.Add(new Record(int.Parse(line_split[1].Trim()), line_split[0]));
        }
        records.Sort(); // Los ordena
        // Si hay menos de 10 scores, se inserta
        if (records.Count < 10)
            records.Add(new Record(PlayerPrefs.GetInt("Puntos"), player_name));
        else
        // Si hay mas de 10, se comprueba si tiene más puntos que el último
        // y se sustituye si es cierto.
            if (PlayerPrefs.GetInt("Puntos") > records[0].getPuntos())
                records.Insert(records.Count - 1,
                               new Record(PlayerPrefs.GetInt("Puntos"),
                               player_name));
```

```
        // Se reordena
        records.Sort();
        records.Reverse(); // Se le da la vuelta para ordenarlo de mayor a menor.
        // Escribe en el fichero los puntos.
        HandleText.WriteString("score",records[0].ToString(),false);
        for (int i = 1; i < records.Count; i++)
            HandleText.WriteString("score",records[i].ToString(),true);

    }
```

3. Fondo del Juego

Para generar el fondo se han utilizado dos técnicas diferentes. Una para generar el fondo a medida que el jugador va avanzando por el mapa y otra para que se mueva el fondo mientras el jugador avanza, dando una perspectiva de 2.5D.

3.1. Tiling

El tiling es una técnica utilizada en el diseño de fondos de videojuegos en la que se divide una imagen en pequeñas secciones (llamadas "tiling") que se pueden repetir y unir sin dejar notar la repetición, lo que permite crear fondos y escenarios detallados y dinámicos con una menor carga en los recursos del sistema.

La técnica de Tilling es ampliamente utilizada en videojuegos de 2D, donde los fondos pueden estar formados por múltiples capas de tilings, creando profundidad y dimensión en la imagen.

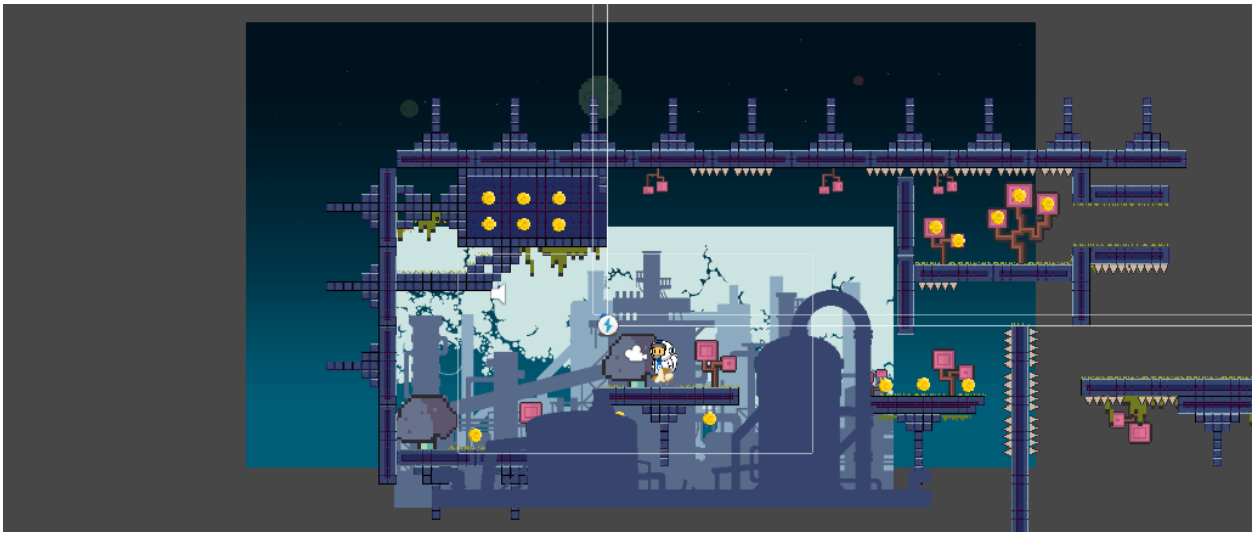

Figura 1: Imagen del videojuego sin utilizar la técnica de tiling

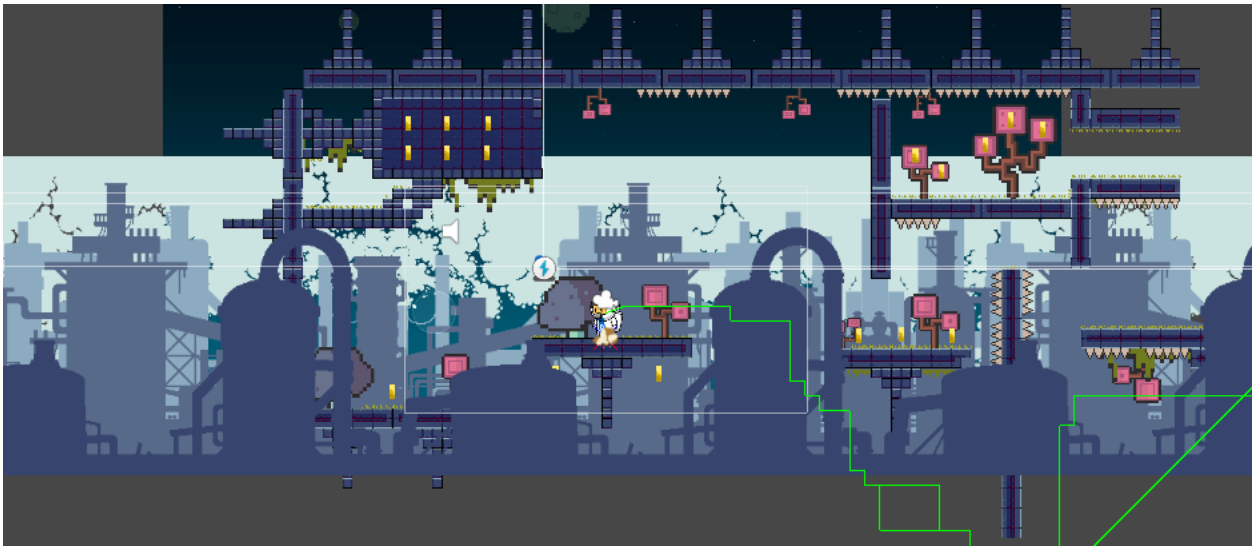

Figura 2: Imagen del videojuego utilizando la técnica de tiling

3.2. Parallaxing

El parallaxing es una técnica utilizada en el diseño de fondos en videojuegos que permite crear la ilusión de profundidad y movimiento en un escenario de 2D o 3D. Se basa en el principio de que los objetos más cercanos parecen moverse más rápido que los objetos más lejanos cuando se observan desde un punto de vista en movimiento.

En el contexto de los videojuegos, el parallaxing se logra al crear varias capas de fondos con diferentes velocidades de desplazamiento. La capa más cercana al jugador se mueve a una velocidad mayor que la capa más lejana, lo que crea una sensación de profundidad y perspectiva en el fondo.

Las diferentes capas que se han utilizado se pueden ver en la siguiente imagen:

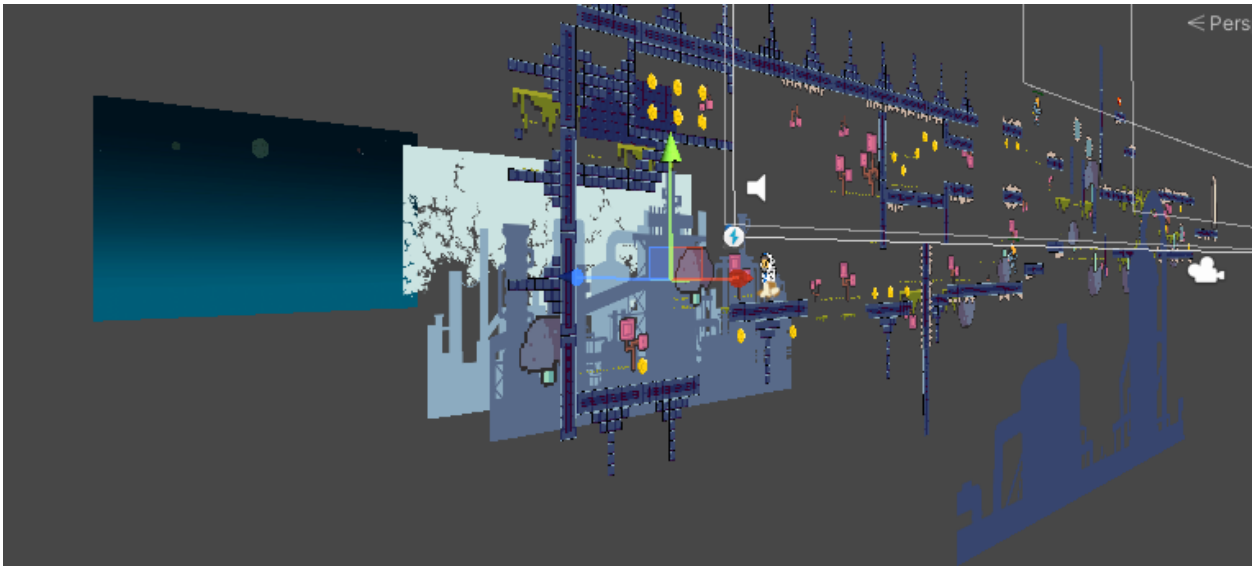

Figura 3: Orden de las capas utilizando parallaxing

4. Implementación de Interfaces

4.1. Menú de Pausa

A parte del menú principal, que no se ha incluido en esta memoria por ser demasiado simple, también se ha incluido un menú de pausa en el juego que se activa pulsando la tecla «ESC».

Este menú permite pausar el juego, reanudarlo o salir de él guardando los puntos y vida del jugador.

Figura 4: Menú de pausa del juego.

4.2. Indicador de vida

Tanto para indicar la vida que tiene el jugador como la de los enemigos se ha utilizado una barra horizontal. En caso de la barra de vida del jugador se sitúa estática arriba a la derecha de la pantalla.

Por otro lado, la de los enemigos se encuentra justo encima de ellos y se mueve con ellos. Esto se ha conseguido utilizando un canvas como hijo del objeto enemigo. De esta forma, la posición del indicador es siempre relativa a la del enemigo.

Figura 5: La barra roja arriba a la derecha indica la salud del jugador, mientras que las barras verdes de encima de los enemigos representa su salud.

5. Inteligencia Artificial

5.1. Búsqueda de Caminos

5.1.1. A*

El algoritmo A* es un algoritmo de búsqueda de camino comúnmente utilizado en videojuegos para encontrar el camino más corto entre dos puntos en un mapa o nivel de juego.
La implementación de este algoritmo en el juego se ha hecho mediante un asset de la página de Aron Grangberg que permite configurar muchos parámetros del mismo.

Este algoritmo se ha implementado en el movimiento de los enemigos voladores debido a su capacidad de moverse por el mapa sin ninguna limitación.

5.1.2. Breadcrumb Pathfinding

En el videojuego se ha intentado implementar la técnica de búsqueda de caminos de las migas de pan, que consiste en dejar un rastro de objetos invisibles para que algún enemigo pueda seguirlas. En este caso se han utilizado cuadrados azules para representar las «migas de pan» en la escena:

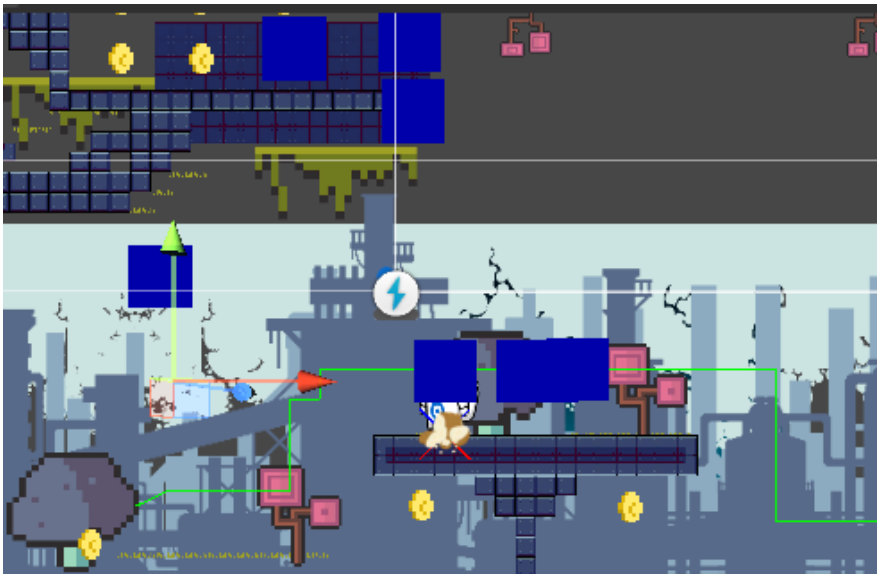

Figura 6: Los cuadrados azules corresponden con las migas de pan de la técnica del mismo nombre. La línea verde corresponde con el camino elegido por un enemigo volador que se encontraba más adelante.

En un principio se intentó hacer con un *asset* descargado de la Asset Store, sin embargo, solo estaba disponible para videojuegos en tres dimensiones. Opté, por tanto, en modificar el código y adaptarlo un poco a las dos dimensiones. Todo funcionaba bien, el enemigo seguía las migas de pan, sin embargo, en algunas ocasiones se quedaba atascado o se quedaba saltando intentando alcanzar alguna de ellas que estaba demasiado alta.

Debido a que no funcionaba correctamente, decidí cambiar la inteligencia de los enemigos terrestres para que simplemente, en lugar de seguir las migas de pan, siguiesen al jugador.

6. Sugerencias Implementadas

En la presentación del juego se recomendó implementar el sonido también al golpear un enemigo para proporcionarle mayor *feedback*. Se han implementado dos nuevos sonidos:

1. Sonido de golpe a un enemigo.
2. Sonido de muerte de un enemigo.

7. Recursos

Para esta práctica se han utilizado los siguientes recursos para descargar assets tales como sprites, sonidos o mecánicas.

- **Diferentes sprites**: https://assetstore.unity.com/
- **Asset del algoritmo A***: http://www.arongranberg.com
- **Diferentes sonidos**: https://www.pond5.com/es/