Sistema de Guardado Básico en Unity

Triana Casal Rafael Alberto

Introducción:

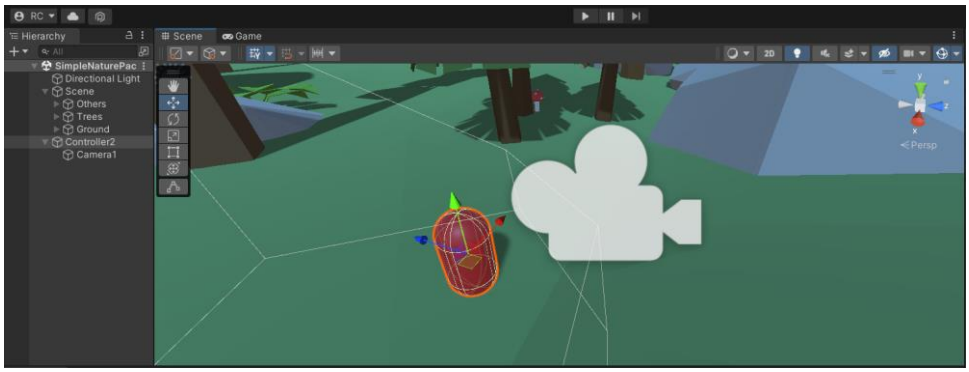
Este es un sistema de guardado hecho en Unity3D, con un código hecho en Visual Studio (Cabe aclarar que la versión que yo ocupo es Unity 2021.3.10f1 y *VisualStudio* 2022). En este documento tendrás una breve explicación acerca del código, assests utilizados y cualquier movimiento que haya hecho dentro del mismo programa.

- *Numero 1: Elementos de Escena*

Aquí mostraré todo lo utilizado dentro de la escena y de donde ha sido obtenido.

Personaje:

Para el personaje he utilizado una figura 3D básica del mismo Unity; el material utilizado es completamente opcional, sin embargo, si buscas asignarle uno tendrás que hacerlo desde el *click* derecho en *assets*, crear ---> material, para luego asignarlo a donde lo quieres.



Propiedades del personaje (Player/Controller2):

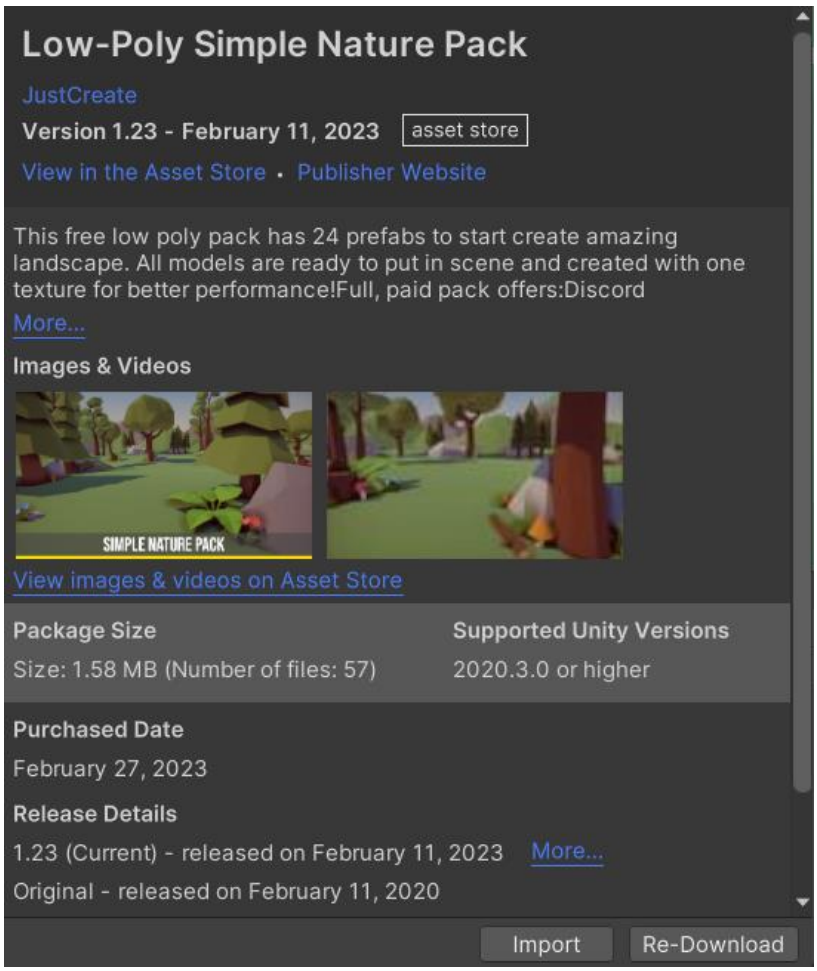
A la píldora que ha sido introducida a nuestra escena, se tendrán que agregar los siguientes componentes.

En este caso ya contamos con un código de control de movimiento de personaje en primera persona que se encuentra ya dentro de los scripts llamado *"Controller2"*. Si se llegase a contar con alguna duda acerca de este código se podrá revisar el contenido de este para cualquier aclaración ya que cada línea de código cuenta con su respectiva explicación como un comentario añadido o también buscar la documentación de este.

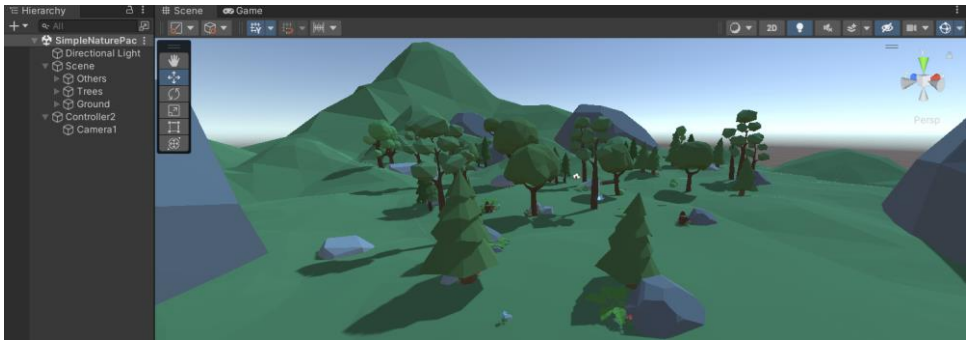
Mapa:

Para el mapa he utilizado un paquete gratuito de Unity que se puede descargar directamente desde el apartado de *window* ----> *Asset Store* y te dará lo opción de descargarlo para poder utilizar algunas figuras o

materiales que Unity no cuenta de base. Sin embargo, se podrá trabajar con todas las opciones que Unity ya nos da de sus objetos bases.



Este es el paquete en específico utilizado, donde es nada meramente visual para la construcción del mapa.



Todo elemento del mapa esta con una *Box Collider* donde he modificado el mismo *collider* a mi gusto. Esto es para poder darles propiedades físicas a los objetos y nuestro personaje no pueda traspasarlos.

- *Número 2: Códigos (Scripts):*

Para este sistema de guardado se han ocupado tres códigos distintos, los cuales cada uno cuenta con una función en específico ya sea la de almacenar los datos, hacer una función de "liga" entre *srcipts y* entre otras cosas. Dentro del código tendrás agregado comentarios explicando las líneas de código y su funcionalidad.

Como recomendación haz una carpeta específica para cada elemento que utilizarás, una para código, otra para materiales, prefaps, etc.

Datos a Guardar (PlayerData):

En este *script* se colocarán los datos que se buscarán guardar; en esta ocasión para la demostración solo se ocuparán solamente los datos de la locación de nuestro jugador sin embargo se podrán guardar otros tipos de datos como pude ser un puntaje, parámetros de vida, energía, entre otras cosas que uno decida.

```
public PlayerData (Controller2 controller2)
    //Aquí se toma de parámetro del jugador, en este caso llamado "controller2"
{
    position[0] = controller2.transform.position.x;
    position[1] = controller2.transform.position.y;
    position[2] = controller2.transform.position.z;
    //Este apartado guardará la posición del jugador (Aquí dependerá el tipo de juego si es 2D o 3D, ya que se gaurdarán las coordenadas)
}
```

Como se puede observar cada línea de código tiene su comentario respectivo para la aclaración de funcionalidad del código.

position[0] = controller2.transform.position.x;

position[1] = controller2.transform.position.y;

position[2] = controller2.transform.position.z;

Las propiedades de estas variables son para la localización a base de coordenadas de nuestro jugador. Esto dependerá de si el juego es 2D o 3D para saber que coordenadas son las que se utilizarán. En este caso son tres (Eje X, Y y Z).

Para esta parte del código se utilizará de atributo de *System.Serilizable* esto con el fin de poder serializar los datos.

```
[System.Serializable]
//Habilitará la serialización de datos de esta clase
```

Guardado y Carga de Datos (SaveManager):

Este apartado se encargará de hacer el código específico de funcionalidad del guardado y carga de datos, de a donde serán dirigidos y almacenados.

Para este código se utilizará dos *NameSpaces*, en particular el *UnityEngine* y *System.IO*, donde nos permitirá trabajar con archivos y convertir datos en binario y poder almacenarlos en archivos respectivamente la base.

```
using UnityEngine;
using System.IO;
//Permitirá trabajar con archivos
using System.Runtime.Serialization.Formatters.Binary;
//Permitirá convertir los datos en binario para guardarlos en archivo
```

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

```
public static class SaveManager
//En este código se darán las acciones para que se puedan salvar los datos y poder cargarlos
{
```

Este apartado se declarará *static* para que no sea una clase instanciada si no que cargará y guardará datos de otros *scripts*.

Dentro de este se encontrará toda la parte del programa que se encargará de direccionar todos los datos que buscamos guardar y el lugar donde estos terminarán.

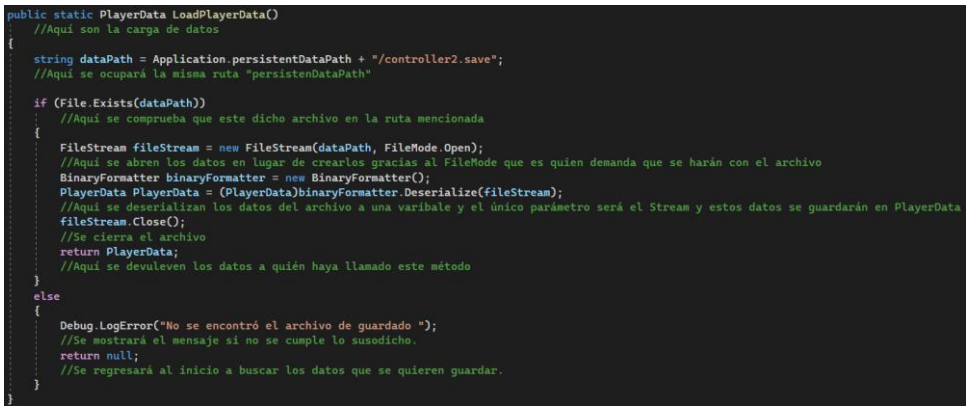
Para este apartado se ocupará el parámetro de *Controller2* el cual en este caso termina siendo el *personaje* o *player* y tomará los datos que hemos puesto ateriormente en nuestro código de *playerdata*y tomará los datos que hemos puesto anteriormente en nuestro código de *playerdata*. Una vez hecho esto se definirá la ruta que tomarán estos datos que serán convertidos en binarios para su almacenamiento y lugar de destino

Una vez este todo esto definido no se puede olvidar el cerrar el archivo.

```
public static void SavePlayerData(Controller2 controller2)
    {
        PlayerData playerData = new PlayerData(controller2);
            string dataPath = Application.persistentDataPath +
"/controller2.save";
        FileStream fileStream = new FileStream(dataPath, FileMode.Create);
        BinaryFormatter binaryFormatter = new BinaryFormatter();
        binaryFormatter.Serialize(fileStream, playerData);
        fileStream.Close();
    }
```

Para la segunda parte de este código se habrá hecho casi lo mismo con la diferencia que habrá un apartado que se encargará de “deserializar” los datos en binario lo grándose con la siguiente línea de código:

```
PlayerData PlayerData =
(PlayerData)binaryFormatter.Deserialize(fileStream);
```



```
public static PlayerData LoadPlayerData()
    //Aquí son la carga de datos
{
    string dataPath = Application.persistentDataPath + "/controller2.save";
    //Aquí se ocupará la misma ruta "persistenDataPath"

    if (File.Exists(dataPath))
        //Aquí se comprueba que este dicho archivo en la ruta mencionada
    {
        FileStream fileStream = new FileStream(dataPath, FileMode.Open);
        //Aquí se abren los datos en lugar de crearlos gracias al FileMode que es quien demanda que se harán con el archivo
        BinaryFormatter binaryFormatter = new BinaryFormatter();
        PlayerData PlayerData = (PlayerData)binaryFormatter.Deserialize(fileStream);
        //Aquí se deserializan los datos del archivo a una varibale y el único parámetro será el Stream y estos datos se guardarán en PlayerData
        fileStream.Close();
        //Se cierra el archivo
        return PlayerData;
        //Aquí se devuleven los datos a quién haya llamado este método
    }
    else
    {
        Debug.LogError("No se encontró el archivo de guardado ");
        //Se mostrará el mensaje si no se cumple lo susodicho.
        return null;
        //Se regresará al inicio a buscar los datos que se quieren guardar.
    }
}
```

Al igual que el anterior no olvidar el cerrar el archivo y ya es algo opcional el que se imprima el mensaje de que se haya o no logrado el guardar datos o el cargarlos.

```
public static PlayerData LoadPlayerData()
  {
    string dataPath = Application.persistentDataPath + "/controller2.save";
    if (File.Exists(dataPath))
    {
      FileStream fileStream = new FileStream(dataPath, FileMode.Open);
       BinaryFormatter binaryFormatter = new BinaryFormatter();
      PlayerData PlayerData =
(PlayerData)binaryFormatter.Deserialize(fileStream);
      fileStream.Close();
      return PlayerData;
    }
    else
    {
      Debug.LogError("No se encontró el archivo de guardado ");
      return null;
```

```
    }
}
```

Comando de Guardado y Carga de Datos (Controller2):

Este Script tiene en su contenido el movimiento del personaje sin embargo cuenta con un apartado hecho con un comentario para diferenciar en donde interviene el guardado y carga de datos, en donde se encargará con una condicional, el asigna un botón para que genere la acción y de cargar datos respectivamente.

```
//Guardado y Carga de datos

if (Input.GetKeyDown(KeyCode.G))
    //Condicional que para presionar la tecla "G" y ocurra el evento que se encunetra dentro de esta.
{
    SaveManager.SavePlayerData(this);
    //Se salvarán los datos si la acción es realizada.
    Debug.Log("Datos Guardados");
    //Se escribirá esto en conosla una vez realizada la acción con éxito.
}

if (Input.GetKeyDown(KeyCode.C))
//Condicional que para presionar la tecla "C" y ocurra el evento que se encunetra dentro de esta.
{
    PlayerData playerData = SaveManager.LoadPlayerData();
    //Los datos se cargarán al relizar la acción
    transform.position = new Vector3(playerData.position[0], playerData.position[1], playerData.position[2]);
    //Los datos de guardado que serán cargados una vez se haga la acción
    Debug.Log("Datos Cargados");
    //Se escribirá esto en conosla una vez realizada la acción con éxito.
}
```

Se puede asignar a otra acción que no sea oprimiendo un botón, sin embargo, en este caso lo he decidido de esta manera para una demostración sencilla.

Número 3: Asignación

En esta ocasión no existe ningún tipo de asignación más que la que ya está dada al personaje con su respectivo código *Controller2*. Esto es gracias a que entre los códigos dados se llaman entre si formando un enlace de datos.

Recomendaciones:

- Recuerda que muchas veces dependerá de la versión de Unity o Visual Studio que uno utilice.
- Siempre asegúrate de una correcta escritura de los comandos utilizados, variables y valores lógicos.

- Prioriza el orden dentro de tus proyectos, esto para una mejor presentación y una segura optimización del tiempo.
- No olvides cerra las líneas de código con: ; / () / {} / [] respectivamente.
- Recuerda guardar tus progresos siempre que los actualices o hagas un cambio, ya sea desde el menú de archivo o con el *shorcut: ctrl + s*, ya sea el código como la escena.
- No olvidar cerrar los archivos o *streams* con: *fileStream.Close();*
- Respeta siempre la ortografía dada a variables