**Implementación de manejo de archivos para carga de información modificable en aplicaciones en Unity**

**Universidad Nacional Autónoma de México**


**Dirección General de Cómputo y de Tecnologías de información y Comunicación**

**Implementación de manejo de archivos para carga de información modificable en aplicaciones de Unity**

**[Caso Barrio Universitario y Protozoarios sistémicos]**

**Tayde Martín Cruz Lovera**

**UNAM DGTIC DVRV**

**Ciudad Universitaria, 2022.**

**Resumen**

El uso de archivos de configuración en Unity permite almacenar información modificable y en algunos casos, incluso sin tener que recompilar la aplicación, es una práctica común debido a la facilidad de uso y flexibilidad de manejar información para diversos fines dentro de una aplicación.

**Introducción**

En Unity es posible almacenar información para diversos usos, dependiendo de la complejidad de la aplicación es posible encontrarse con las siguientes necesidades en las cuales los archivos de configuración pueden ser la solución:

1. Almacenamiento de información del usuario, por ejemplo, nombre de usuario, puntos o resultados de evaluaciones, estadísticas, progreso, niveles o ejercicios resueltos, etc.
2. Almacenamiento de configuraciones personalizadas por el usuario, por ejemplo, nivel de volumen de los sonidos de la aplicación, tamaño de fuente de los textos, nivel de calidad de los gráficos de la aplicación, idioma, etc.
3. Perfiles de usuarios, en los casos en los que la aplicación este pensada para ser usada por distintos usuarios en un mismo dispositivo.
4. Almacenamiento de información textual de la aplicación.
5. Aplicaciones con la opción de mostrar la información en varios idiomas.
6. Intercambio de datos de la aplicación con servidores externos.
7. Creación dinámica de escenas o niveles en tiempo de ejecución.

En cualquiera de los casos anteriores, el uso de archivos de configuración es una solución fácil de integrar dentro del desarrollo de aplicaciones en Unity.

**Marco teórico**

Hay varias formas de crear y utilizar archivos de configuración en Unity, las principales son:

- PlayerPrefs
- TXT
- XML
- JSON

PlayerPrefs

Clase de Unity que se utiliza para guardar y recuperar información persistente en el dispositivo, por lo mismo, la información se guarda directamente en los archivos del dispositivo en donde se haya instalado y ejecutado la aplicación. Es importante señalar que los datos no se almacenan en un servidor o en la nube, al momento en que el usuario borre o desinstale la aplicación los valores almacenados se perderán. Además, es importante considerar que los datos almacenados no son encriptados, por lo que pueden ser fácilmente modificados por el usuario, por lo cual, no se recomienda su uso para almacenar información confidencial.

Se utiliza para guardar y leer información entre sesiones de juego o de uso de la aplicación, es decir, permite almacenar información importante antes de cerrar la aplicación y recuperarla en la siguiente ejecución.

Permite almacenar valores como enteros, flotantes o cadenas de texto. La clase proporciona una serie de métodos para guardar y recuperar los valores, por ejemplo:

- SetInt
- GetInt
- SetFloat
- GetFloat
- SetString
- GetString

El uso de la clase es muy fácil e intuitivo. Por ejemplo:

```
// Guardar un valor entero
PlayerPrefs.SetInt("AlgunEntero", 100);

// Recuperar el valor entero almacenado previamente
int algunEntero = PlayerPrefs.GetInt("AlgunEntero");
```

Archivos de texto plano

Existen varias clases que permiten realizar la lectura y escritura de archivos TXT en Unity, las principales son File, StreamReader y StreamWriter. Los archivos tienen que estar almacenados en el dispositivo y en algunos casos es necesario considerar si se necesita tener permisos para el acceso a los archivos, dependiendo de la plataforma en donde se esté ejecutando la aplicación.

Ejemplo de uso de la clase File:

```
// Lectura de un archivo
string filePath = "path/to/file.txt";
string fileContent = File.ReadAllText(filePath);
string[] lines = fileContent.Split('\n');

// Escritura de un archivo
string filePath = "path/to/file.txt";
string textToWrite = "Contenido del archivo de texto";
File.WriteAllText(filePath, textToWrite);
```

Ejemplo de uso de la clase StreamReader:

```
// Lectura de un archivo
string filePath = "path/to/file.txt";
using (StreamReader reader = new StreamReader(filePath))
{
    string fileContent = reader.ReadToEnd();
}
```

Ejemplo de uso de la clase StreamWriter:

```
// Escritura de un archivo
string filePath = "path/to/file.txt";
string textToWrite = "Contenido del archivo de texto";
using (StreamWriter writer = new StreamWriter(filePath))
{
```

```
    writer.WriteLine(textToWrite);
}
```

XML

Es un lenguaje de etiquetas utilizado para almacenar y transmitir datos estructurados, tiene una sintaxis similar a HTML en donde se define la estructura de los datos usando etiquetas.

Unity, por medio de C#, proporciona una forma nativa para la lectura y escritura de archivos XML por medio de la clase XmlSerializer.

Ejemplo de uso de la clase XmlSerializer.

```
using System.Xml.Serialization;
using System.IO;

// Clase para almacenar los datos del archivo XML
public class AppData
{
    public string usuario;
    public int puntos;
}

public class XMLManager
{
    // Método para guardar los datos en un archivo XML
    public void SaveData(AppData data)
    {
        XmlSerializer serializer = new XmlSerializer(typeof(AppData));
        using (StreamWriter stream = new StreamWriter(Application.dataPath +
"/archivoXML.xml"))
        {
            serializer.Serialize(stream, data);
        }
    }

    // Método para cargar los datos desde un archivo XML
    public AppData LoadData()
    {
        XmlSerializer serializer = new XmlSerializer(typeof(AppData));
        using (StreamReader stream = new StreamReader(Application.dataPath +
"/archivoXML.xml"))
        {
            return serializer.Deserialize(stream) as AppData;
        }
    }
}
```

Es importante asegurarse de que el archivo archivoXML. xml exista, Application.dataPath es la ruta a la carpeta de datos de la aplicación en el dispositivo de ejecución.

JSON

Es un formato de intercambio de datos basado en texto plano, se utiliza ampliamente en aplicaciones web o móviles y en Unity es posible su uso. Unity proporciona la clase JsonUtility para la serialización y deserialización de archivos JSON.

Ejemplo de lectura y escritura de archivos JSON en Unity:

```
using System.IO;
using UnityEngine;

// Clase para almacenar los datos del archivo JSON
public class AppData
{
    public string usuario;
    public int puntos;
}

public class JSONManager
{
    // Método para guardar los datos en un archivo JSON
    public void SaveData(AppData data)
    {
        string json = JsonUtility.ToJson(data);
        File.WriteAllText(Application.dataPath + "/archivoJSON.json", json);
    }

    // Método para cargar los datos desde un archivo JSON
    public AppData LoadData()
    {
        string json = File.ReadAllText(Application.dataPath + "/archivoJSON.json");
        return JsonUtility.FromJson<AppData>(json);
    }
}
```

Es importante asegurarse de que el archivo archivoXML. xml exista, Application.dataPath es la ruta a la carpeta de datos de la aplicación en el dispositivo de ejecución.

También es posible usar librerías de terceros para implementar la lectura y escritura de archivos JSON en Unity.

Las ventajas del uso de archivos JSON son las siguientes:

- Es un formato de texto plano, por lo que es fácil su lectura, entendimiento y modificación.
- Tiene un formato estructurado que permite una organización en jerarquía.
- Es ampliamente compatible con múltiples lenguajes de programación y herramientas de desarrollo, incluso con Unity.
- Permite una forma sencilla de serializar objetos, es decir, convertir un objeto en una representación de datos para ser almacenado.
- Es un formato ligero, que no ocupa mucho espacio de almacenamiento.
- Soporta múltiples tipos de datos para almacenar.

**Método**

Para este documento se hará uso de archivos JSON para el almacenamiento de información, el ejemplo práctico consistirá en la lectura de información textual para ser mostrado en una serie de ventanas de una interfaz de usuario sencilla dentro de una aplicación.

Lo primero que necesitamos es crear un archivo JSON por cada ventana de la información requerida para ser mostrada en la aplicación, para este ejemplo usaremos 3 ventanas y 3 archivos. Es posible usar arreglos en un archivo JSON, pero la clase nativa de Unity no soporta la deserialización de los mismos, para ello se necesitaría usar código adicional o una librería de terceros.

Archivo ArchivoDatos1.json

```
{
  "titulo": "Ventana 1",
  "informacion": "Algún contenido para ser mostrado en la ventana 1."
}
```

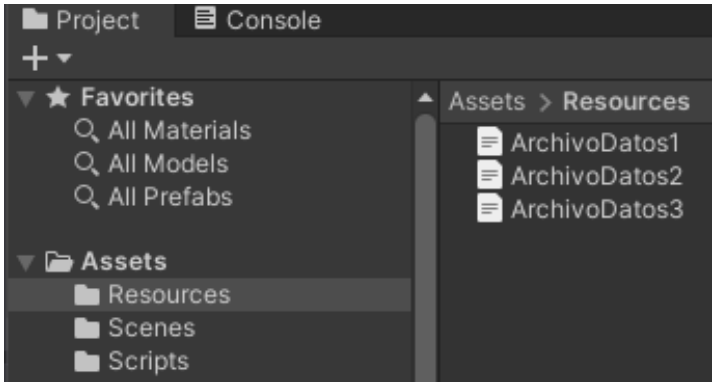
Archivo ArchivoDatos2.json

```
{
  "titulo": "Ventana 2",
  "informacion": "Contenido para ser mostrado en la otra ventana, la 2."
}
```

Archivo ArchivoDatos3.json

```
{
  "titulo": "Ventana 3",
  "informacion": "Otro contenido para ser mostrado en la ventana 3."
}
```

El archivo lo guardaremos en la carpeta Resources del proyecto de Unity, si la carpeta Resources no existe dentro de la carpeta Assets del proyecto, es necesario crearla. Es una carpeta especial de Unity utilizada para cargar recursos en tiempo de ejecución sin importar la plataforma destino de la aplicación, el contenido de esta carpeta se incluye en el paquete de la aplicación al momento de la compilación sin importar que no estén referenciados en una de las escenas a compilar, por lo que es importante tener cuidado de que los archivos que sean almacenado dentro de ella realmente sean usados en la aplicación.



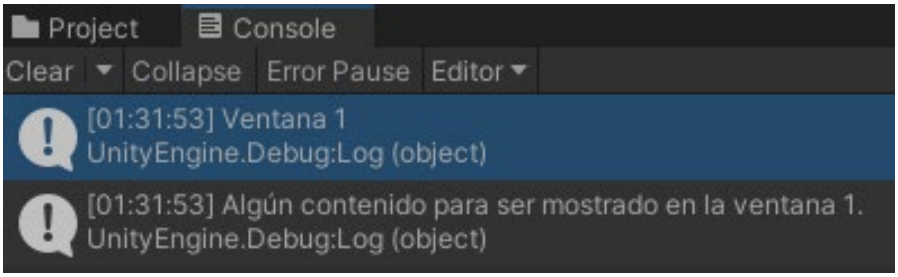

Crearemos un script que permite leer los 3 archivos JSON, deserializarlos y almacenarlos en un arreglo.

Script LecturaJson.cs

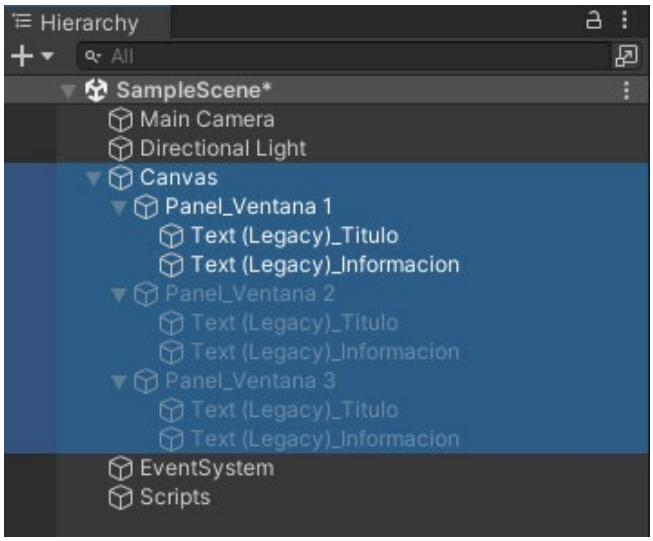
```
using UnityEngine;
```

```
public class LecturaJson : MonoBehaviour
{
    Public ClaseDatosJSON[] objetos = new ClaseDatosJSON[3];

    // Realizamos la lectura de los archivos JSON e imprimimos en la consola el
resultado de la deserialización del ArchivoDatos1 como prueba
    void Awake()
    {
        TextAsset jsonFile = Resources.Load<TextAsset>("ArchivoDatos1");
        objetos[0] = JsonUtility.FromJson<ClaseDatosJSON>(jsonFile.text);

        jsonFile = Resources.Load<TextAsset>("ArchivoDatos2");
        objetos[1] = JsonUtility.FromJson<ClaseDatosJSON>(jsonFile.text);

        jsonFile = Resources.Load<TextAsset>("ArchivoDatos3");
        objetos[2] = JsonUtility.FromJson<ClaseDatosJSON>(jsonFile.text);

        Debug.Log(objetos[0].titulo);
        Debug.Log(objetos[0].informacion);
    }
}

// Clase para almacenar los datos del archivo JSON
[System.Serializable]
public class ClaseDatosJSON
{
    public string titulo;
    public string informacion;
}
```

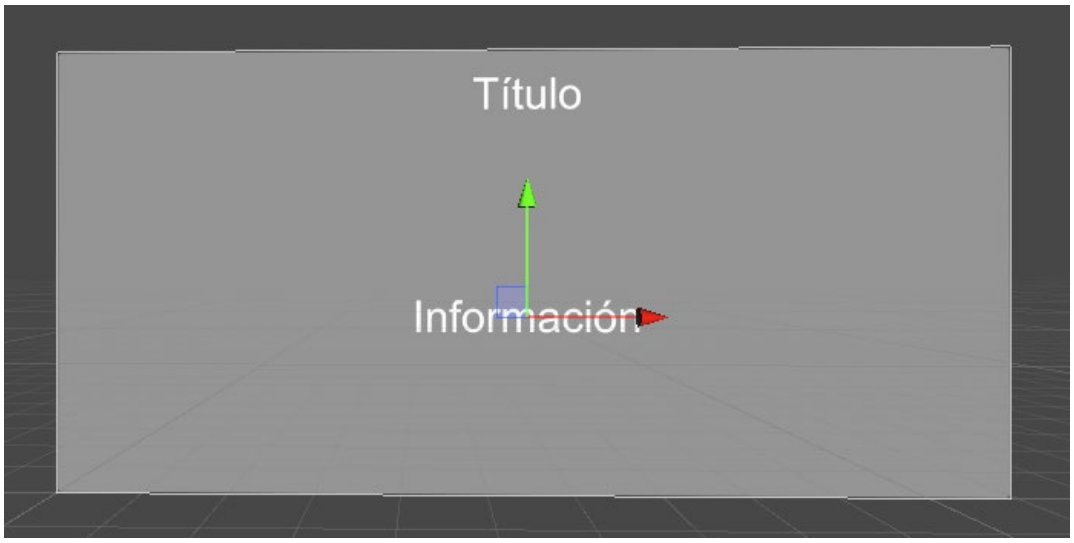
Agregamos el script creado a algún GameObject en la escena y ejecutamos la aplicación en el Editor dando click en el botón Play. En la consola de Unity debería aparecer el contenido leído del JSON de la siguiente forma:




De esta forma se confirma que la lectura se ha realizado de forma correcta. El siguiente paso es crear la interfaz de usuario con 3 ventanas donde se mostrará la información leída de los 3 archivos JSON, la cual quedaría de la siguiente forma:




Dejando deshabilitados las ventanas 2 y 3, y dejando habilitada únicamente la ventana 1 para que no se sobrepongan.



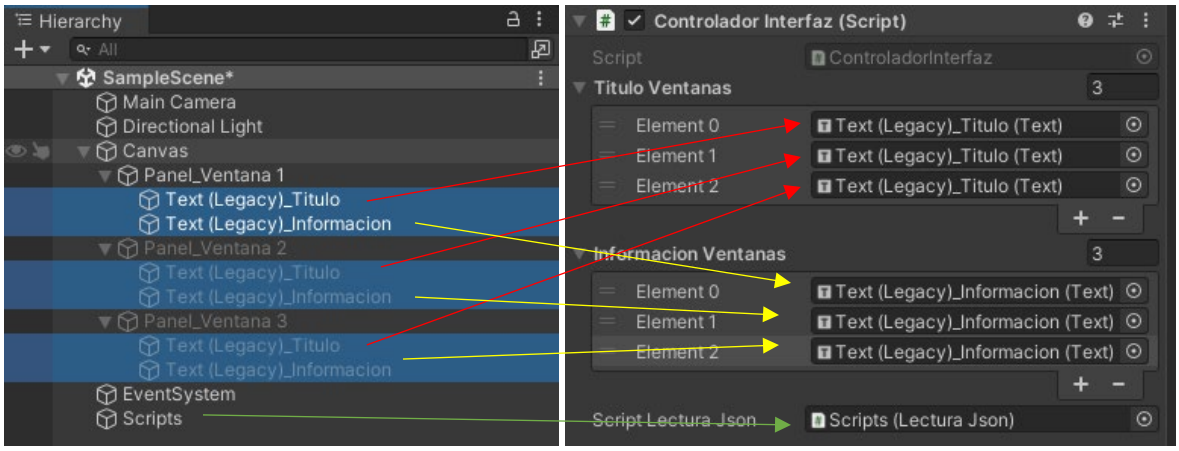

Ahora crearemos otro script para cargar la información leída de los archivos JSON en cada una de las ventanas de la interfaz de usuario.

Script ControladorInterfaz.cs

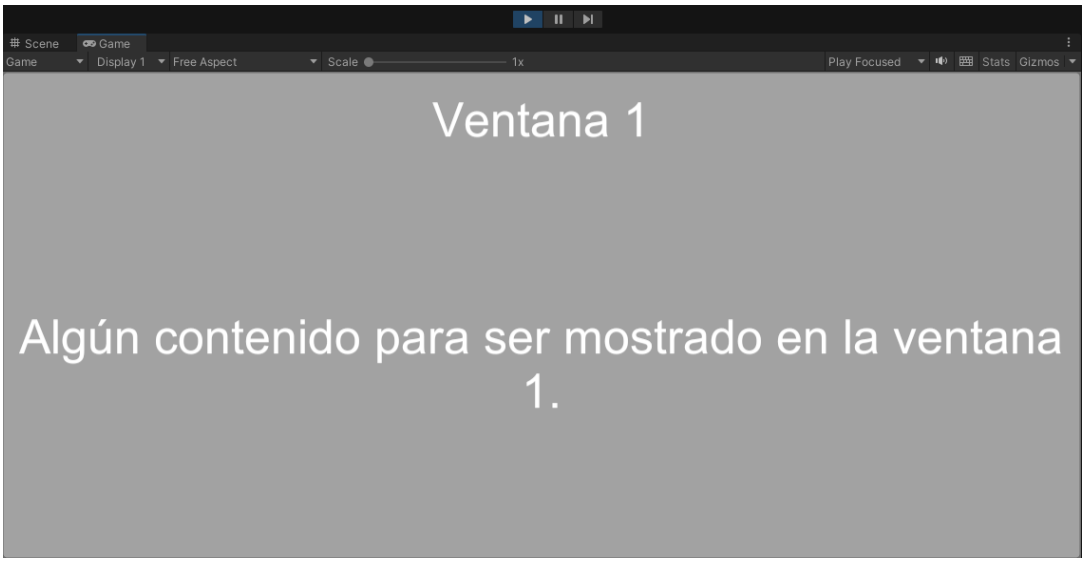
```
using UnityEngine;
using UnityEngine.UI;

public class ControladorInterfaz : MonoBehaviour
{
    // Referencia, para ser asignada en el Inspector, a los 3 elementos Text de
las ventanas correspondientes a los títulos
    public Text[] tituloVentanas;
    // Referencia, para ser asignada en el Inspector, a los 3 elementos Text de
las ventanas correspondientes a la información
    public Text[] informacionVentanas;
```

```
    // Referencia, para ser asignada en el Inspector, al script encargado de leer
los JSON y almacenar la información
    public LecturaJson scriptLecturaJson;

    void Start()
    {
        tituloVentanas[0].text = scriptLecturaJson.objetos[0].titulo;
        tituloVentanas[1].text = scriptLecturaJson.objetos[1].titulo;
        tituloVentanas[2].text = scriptLecturaJson.objetos[2].titulo;

        informacionVentanas[0].text = scriptLecturaJson.objetos[0].informacion;
        informacionVentanas[1].text = scriptLecturaJson.objetos[1].informacion;
        informacionVentanas[2].text = scriptLecturaJson.objetos[2].informacion;
    }
}
```

Agregamos el script a algún GameObject de la escena y asignamos las correspondientes referencias de la siguiente forma:




Si ejecutamos la aplicación, el resultado será el siguiente:

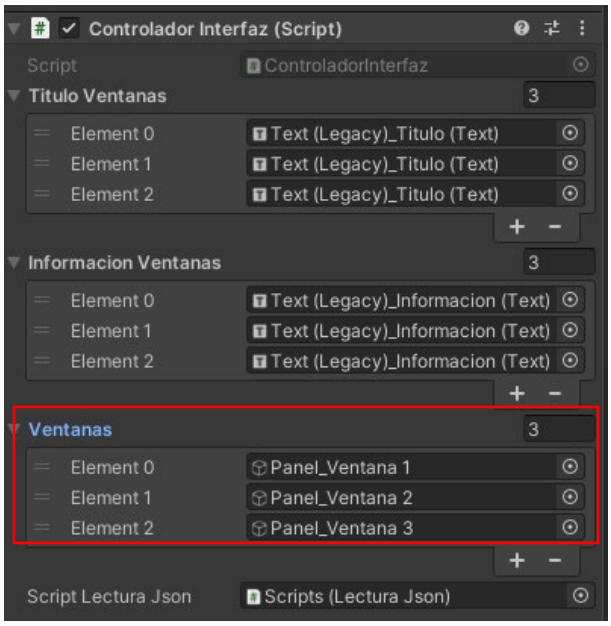



Únicamente resta un botón para cambiar entre ventanas y escribir el código necesario para realizar ese comportamiento.

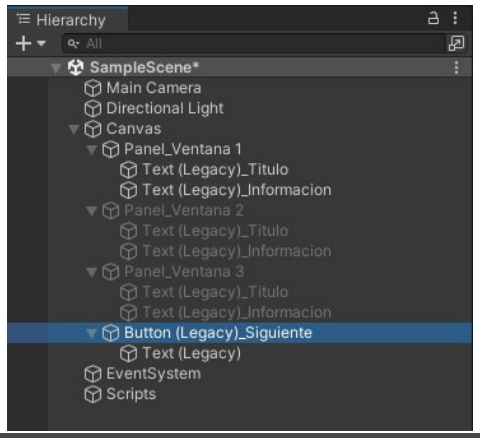
Script ControladorInterfaz.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ControladorInterfaz : MonoBehaviour
{
    // Referencia, para ser asignada en el Inspector, a los 3 elementos Text de
las ventanas correspondientes a los títulos
    public Text[] tituloVentanas;
    // Referencia, para ser asignada en el Inspector, a los 3 elementos Text de
las ventanas correspondientes a la información
    public Text[] informacionVentanas;

    // Referencia, para ser asignada en el Inspector, a los 3 grupos de
GameObjects correspondientes a cada ventana
    private GameObject[] ventanas = new GameObject[3];

    // Referencia, para ser asignada en el Inspector, al script encargado de leer
los JSON y almacenar la información
    public LecturaJson scriptLecturaJson;

    private int ventanaActiva = 0;

    void Start()
    {
        tituloVentanas[0].text = scriptLecturaJson.objetos[0].titulo;
        tituloVentanas[1].text = scriptLecturaJson.objetos[1].titulo;
        tituloVentanas[2].text = scriptLecturaJson.objetos[2].titulo;

        informacionVentanas[0].text = scriptLecturaJson.objetos[0].informacion;
        informacionVentanas[1].text = scriptLecturaJson.objetos[1].informacion;
        informacionVentanas[2].text = scriptLecturaJson.objetos[2].informacion;
    }

    public void OnClick_SiguienteVentana()
    {
        ventanas[ventanaActiva].SetActive(false);
        ventanaActiva++;
        if (ventanaActiva == 3)
            ventanaActiva = 0;
        ventanas[ventanaActiva].SetActive(true);
    }
}
```

Asignamos las referencias a los grupos de GameObjects correspondientes a cada ventana:

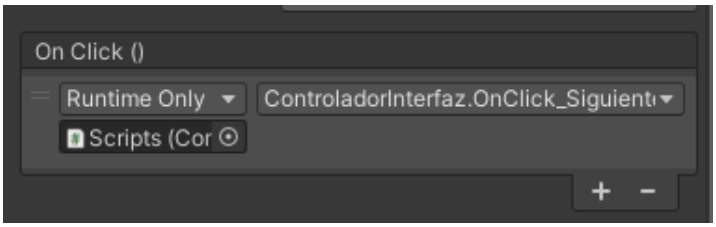



Creamos un botón en nuestra interfaz de usuario:

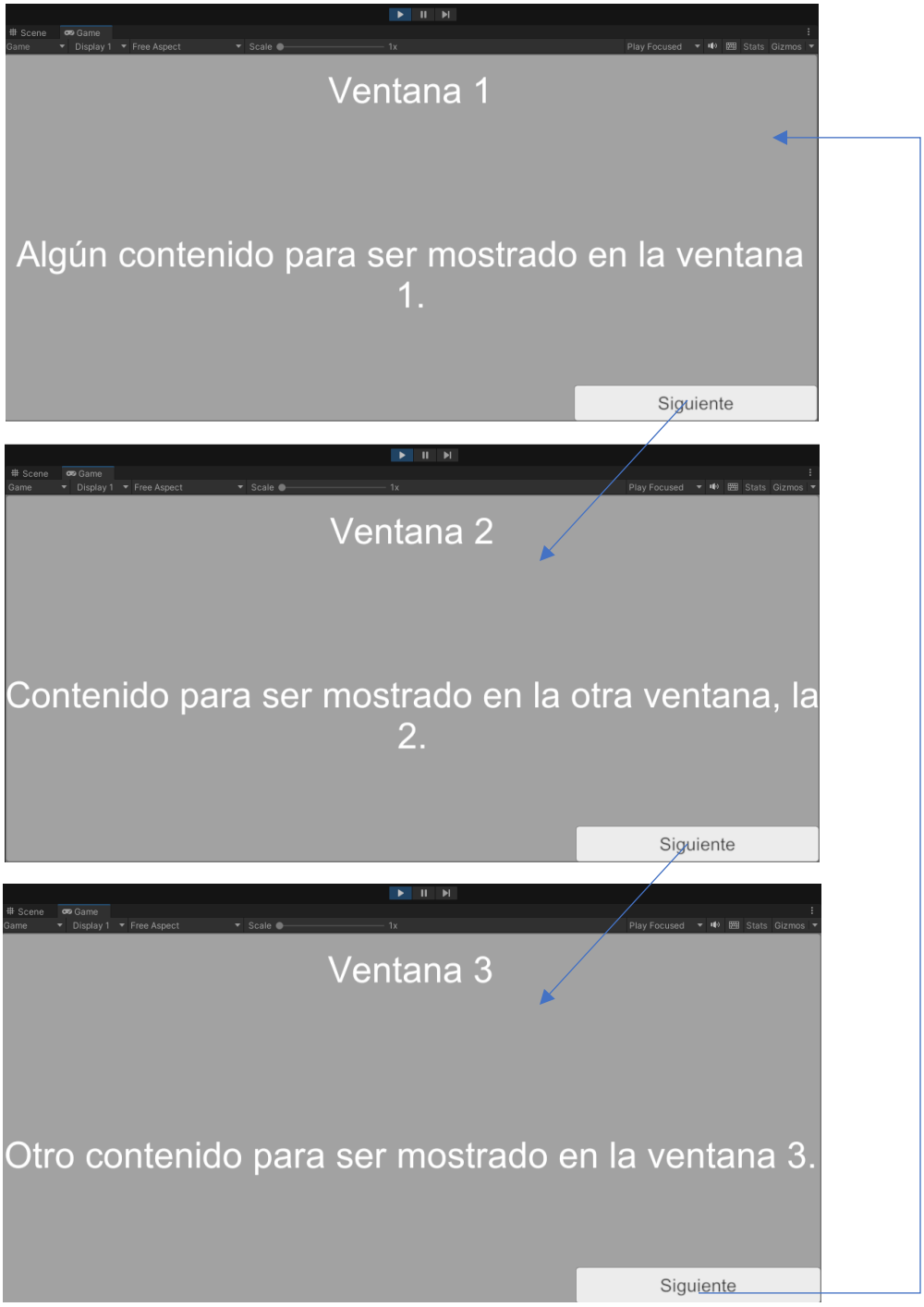




Y asignamos el llamado a la función “OnClick_SiguienteVentana” del script ControladorInterfaz.cs, en el evento OnClick del botón.




Al ejecutar la aplicación, el resultado será el siguiente:



Una de las ventajas de cargar información desde archivos de configuración es la posibilidad de llenar dinámicamente la información de nuestra interfaz de usuario, incluso en tiempo de ejecución, por lo que, en el caso de uso mostrado, realmente la mejor forma de implementarlo sería usando una única ventana y modificando únicamente el texto en el evento OnClick del botón.

Script ControladorInterfaz.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ControladorInterfaz : MonoBehaviour
{
    // Referencia, para ser asignada en el Inspector, al elemento Text de la
ventana correspondiente al título
    public Text tituloVentana;
    // Referencia, para ser asignada en el Inspector, al elemento Text de la
ventana correspondiente a la información
    public Text informacionVentana;

    // Referencia, para ser asignada en el Inspector, al script encargado de leer
los JSON y almacenar la información
    public LecturaJson scriptLecturaJson;

    private int ventanaActiva = 0;

    void Start()
    {
        tituloVentana.text = scriptLecturaJson.objetos[0].titulo;
        informacionVentana.text = scriptLecturaJson.objetos[0].informacion;
    }

    public void OnClick_SiguienteVentana()
    {
        ventanaActiva++;
        if (ventanaActiva == 3)
            ventanaActiva = 0;
        tituloVentana.text = scriptLecturaJson.objetos[ventanaActiva].titulo;
        informacionVentana.text =
scriptLecturaJson.objetos[ventanaActiva].informacion;

    }
}
```

**Resultados y discusión**

Como acabamos de ver, únicamente utilizando archivos JSON y una interfaz de usuario sencilla en Unity, podemos programar una aplicación que se llene de información dinámicamente, fácilmente actualizable o modificable. De la misma forma que se vio anteriormente, hacerlo usando archivos TXT o XML se realiza de una forma similar, en los tres casos, tener la información almacenada en un archivo nos permite modificarla o corregirla fácilmente sin necesidad de trabajar directamente sobre la escena o escenas del proyecto de Unity. Al estar concentrada la información es fácilmente ubicable, un único archivo JSON puede incluir la información de varias escenas o varias ventanas de una interfaz de usuario y para modificarlas bastaría con ubicar el archivo JSON correspondiente y editarlo.

Por el contrario, el uso de PlayerPrefs es más específico, recomendado principalmente para almacenar información temporal entre ejecuciones de la aplicación, ya que la información que puede ser leída, necesariamente se tuvo que haber guardado directamente usando los métodos de la misma clase.

**Conclusiones**

Existe distintas formas de almacenar información en Unity, dependiendo de la plataforma destino, uso e incluso la complejidad de la información a almacenar podemos analizar que opción es la más conveniente, fácil de implementar, y la que da mayores ventajas dependiendo de los objetivos de uso y necesidades específicas de cada proyecto.

La opción de usar JSON para crear archivos de configuración respecto a las alternativas, TXT y XML, es principalmente la facilidad de lectura y comprensión, lo que permite modificar el archivo directamente desde un editor de texto sencillo como el Block de notas de Windows, sin mucho riesgo a equivocarse. Además, por lo general, un archivo JSON es más ligero que un archivo XML y más fácil de serializar que un archivo TXT.

**Trabajo a futuro**

En este documento se exploro la forma de usar JSON como solución para la carga de archivos de configuración dentro del propio proyecto. Una forma de aumentar las ventajas de los archivos de configuración, independientemente si se opta por el uso de JSON, archivos TXT o XML, es el uso de servicios en la nube para actualizar dinámicamente el archivo de configuración de una aplicación, lo cual permitiría la actualización o corrección de información sin necesidad de recompilación, distribución de una actualización ni instalación de una actualización o reinstalación completa de la aplicación por parte del usuario.

Los archivos JSON fácilmente pueden ser almacenados en un servidor, la aplicación sincronizaría la nueva versión del archivo en caso de que exista una versión más actual de la existente en el dispositivo al momento de la comprobación y actualizaría la información correspondiente generada a partir del JSON.

Incluso, si la aplicación esta diseñada para crecer dinámicamente, un archivo JSON actualizado podría permitir que la aplicación tuviera más información, nuevas escenas o ventanas, y por lo general, será necesario también actualizar la base de modelos y texturas de la aplicación.

**Referencias**


- PlayerPrefs. (s.f.) Unity Documentation. https://docs.unity3d.com/ScriptReference/PlayerPrefs.html

- File. (s.f.) Microsoft Documentation https://learn.microsoft.com/en-us/dotnet/api/system.io.file

- XmlSerializer. (s.f.) Microsoft Documentation
  https://learn.microsoft.com/es-es/dotnet/api/system.xml.serialization.xmlserializer

- JsonUtility. (s.f.) Unity Documentation
  https://docs.unity3d.com/ScriptReference/JsonUtility.html

- Resources. (s.f.) Unity Documentation
  https://docs.unity3d.com/ScriptReference/Resources.html

- Application.dataPath. (s.f.) Unity Documentation
  https://docs.unity3d.com/ScriptReference/Application-dataPath.html