VIDEOJUEGOS EN NAVEGADOR

Desarrollo de aplicaciones web basadas en Unity y C# para el proyecto final de Desarrollo de Aplicaciones Web

Francisco Bustillo Barrios

PROYECTO FINAL

DESARROLLO DE APLICACIONES EN UNITY

1. Introducción.

Mi proyecto de fin del ciclo se basa en realizar tres pequeños videojuegos utilizando el motor gráfico Unity unido al lenguaje de desarrollo C#.
Dos de dichos juegos serán insertados en documentos de HTML 5 mediante la API WebGL y el tercero (que será realizado con tecnología VR) será insertado en un documento HTML mediante la API WebVR.

Este proyecto se basa en dos partes, una parte teórica en la que se expondrán las principales características de Unity como medio para desarrollar aplicaciones y una parte práctica en la que expondrán dos videojuegos completamente funcionales y desarrollados con dicha tecnología.

2. ¿Qué es Unity?

Unity es un motor gráfico multiplataforma utilizado para la realización de videojuegos y simuladores tanto en 2 dimensiones como en 3 dimensiones.
Unity también se puede definir como un framework para C# para la creación de aplicaciones que utiliza un sistema de arrastrar y soltar y la simplificación de algunos aspectos del desarrollo de código mediante un editor gráfico.

Un motor gráfico se define como un conjunto de herramientas que ofrecen un motor de renderizado de gráficos, motor de físicas, detector de colisiones, sonidos, administración de memoria y conexiones, entre otras funciones.

VENTAJAS E INCONVENIENTES

Unity ofrece una serie de ventajas notables frente a otros motores gráficos:

- **Unity es de uso gratuito.** Unity es importante tanto como para los nuevos desarrolladores que deseen formarse en desarrollo de videojuegos como para las nuevas empresas que desean entrar en la industria, ya que solo obligan a pagar por su uso cuando se facturan más de 100K dólares.

- **Es multiplataforma.** Unity adapta fácilmente los videojuegos y las aplicaciones a distintos sistemas operativos y sistemas físicos.

- **Es potente.** Unity no solo se usa para juegos indie (de bajo presupuesto, desarrollado por un número reducido de programadores y artistas), sino que también se usa para juegos calificados AAA como Hearthstone o Pokémon Go.

- **Tiene un editor visual.** Tiene un sistema de arrastrar y soltar y puedes compilar el proyecto desde el principio y observar el resultado.

- **Debuggin en directo.** Puedes realizar previsualizaciones de proyecto simplemente pulsando un botón.

- **Asset Store.** Tiene integrada una tienda con contenido tanto de pago como gratuito para integrar en los videojuegos, esto evita a los desarrolladores tener que estudiar diseño gráfico para producir un videojuego o una aplicación.

Por otra parte, Unity también tiene algunos inconvenientes:

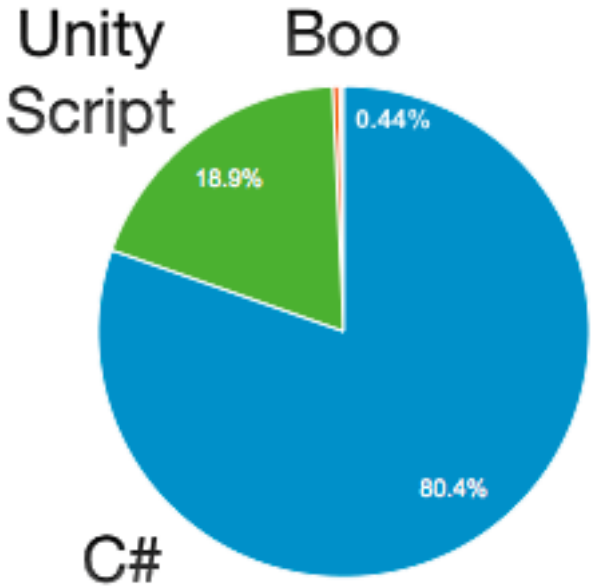
- **Exige tener un equipo para poder realizar un juego original.** Debido a que un videojuego exige tener distintos apartados de especialización como creación de sonidos, montaje de escenarios, creación de modelados 3D etc. Se debe disponer de un equipo de trabajadores para reunir conocimiento en todas estas áreas, ya que para un único desarrollador cubrir todas estas habilidades es muy complicado.

- **Puede ser caro.** Si se requieren características especiales se debe pagar por su uso (como, por ejemplo, evitar que Unity inserte su marca de agua en los juegos antes del comienzo de cada uno o exportar videojuegos a videoconsolas).

LENGUAJES SOPORTADOS

Unity soporta tres lenguajes principales:

C# que es el lenguaje más utilizado en el uso de las aplicaciones.



Unity Script un lenguaje basado en Javascript que es usado como alternativa a C# por su facilidad de uso y aprendizaje. No obstante, este lenguaje posee muchas debilidades debido a que no utiliza las características más avanzadas de Javascript y es considerado como una versión superficial de este lenguaje.

BOO es un lenguaje desarrollado con una sintaxis parecida a Python y adaptado a .NET tratando de simplificar el lenguaje de C# y es un lenguaje muy poco utilizado en Unity. Actualmente se considera no realizar más actualizaciones en el motor gráfico relacionadas con este lenguaje de programación, ya que su falta de uso lo dejará obsoleto.

3. Interfaz Gráfica de Desarrollo

La interfaz de Unity se divide en diversas áreas:

1. **Scene (Escena).** Son las unidades en las que se divide una aplicación (como, por ejemplo, los diferentes niveles de un videojuego, los menús etc.). En cada escena se posicionan todos los elementos de la aplicación.

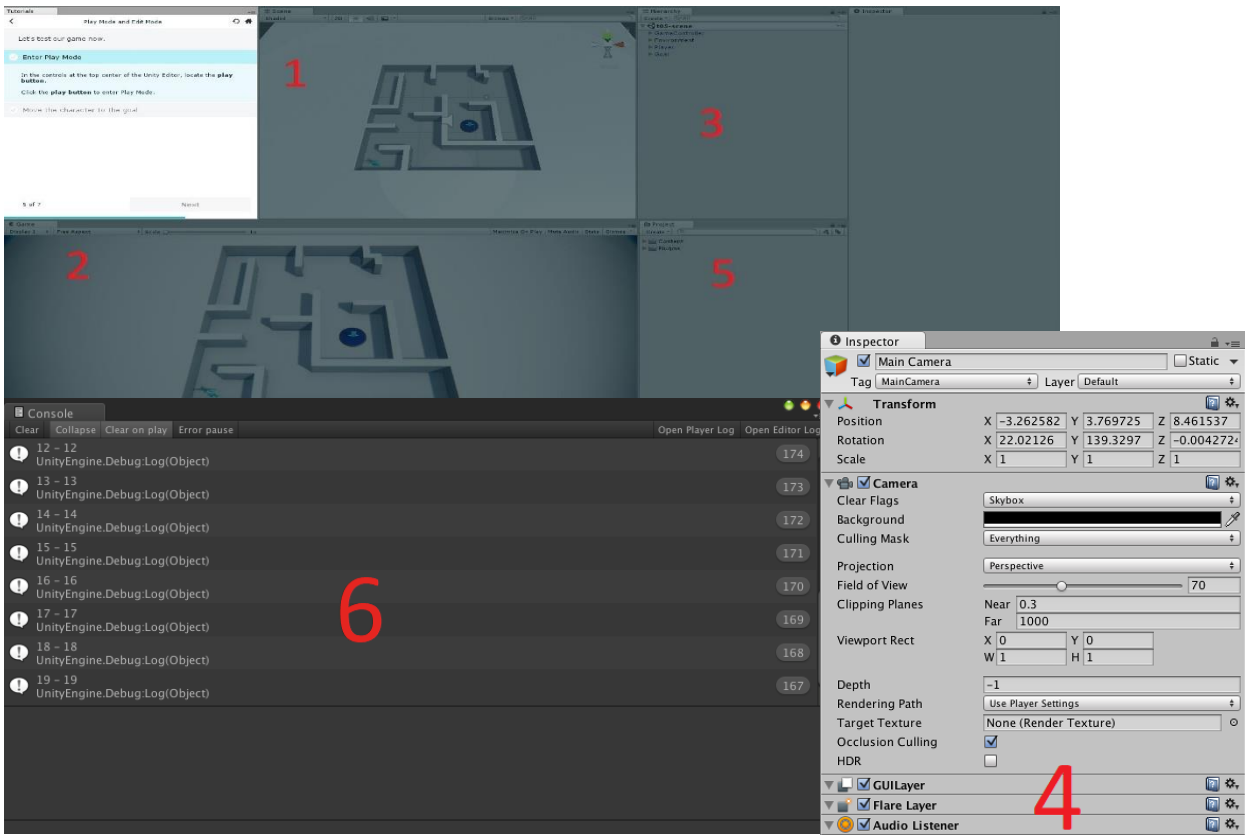
2. **Game (Juego).** Es la parte de la interfaz que muestra la compilación de los diferentes elementos de la aplicación, es decir, es la previsualización del proyecto en el que se trabaja. Al compilar un proyecto, se muestra directamente este elemento.

3. **Hierarchy (Jerarquía).** Es la parte de la interfaz que muestra los Game Objects que se encuentran dentro de una escena. Al seleccionar un objeto en esta parte se muestran sus componentes en el inspector.

4. **Inspector.** Es la parte de la interfaz en la que se modifican los diferentes componentes de un Game Object. Se pueden insertar, eliminar y borrar.

5. **Folders (Carpetas).** Es la parte de la interfaz que nos permite acceder a los elementos y recursos que utilizamos para construir una aplicación. Se debe arrastrar un elemento desde esta parte hasta el hierarchy o el inspector para incluir un elemento en el proyecto.

6. **Console (Consola).** Es la parte de la interfaz que muestra los errores, avisos y mensajes utilizados por Unity o el programador.



4. ¿Qué es un GAME OBJECT?

Los **Game Objects** (GO) son las estructuras más importantes en el desarrollo con Unity. Cada objeto dentro de Unity es considerado un GO.
Un GO se crea en un proyecto de Unity desde el panel de Hierarchy, pero necesita tener propiedades especiales para poder convertirse en un elemento de la aplicación.
(Se asemeja a un objeto creado por una clase, si el objeto está vacío no sirve de nada, pero un objeto que posee funciones y atributos tiene una importancia significativa).

Estas propiedades especiales se conocen como **Componentes**. Dependiendo del GO que se quiera realizar se utilizan distintas combinaciones de componentes. Estos componentes abarcan muchas áreas:

- Malla.
- Efectos.
- Físicas.
- Físicas 2D.
- Audio.
- UI (Interfaz de Usuario).
- Otros.

Hay tres componentes que debemos señalar especialmente: la TRANSFORMACIÓN, la CAMARA y los SCRIPTS.

Al crearse un GO siempre tiene 1 elemento básico: la Transformación.

La transformación es una cualidad que permite localizar un GO en la escena de Unity. Posee tres atributos que son la **posición**, la **rotación** y la **escala**. Estos atributos señalan la posición respecto a la renderización del motor gráfico, el grado de inclinación del objeto y el tamaño del objeto.

El GO más importante es "Camera", sin él no se puede trabajar en Unity. Este GO tiene dos componentes esenciales: **la transformación y la cámara**.

La cámara es un componente que permite la renderización de los objetos en el motor gráfico. Sin este componente no se podría ver ningún objeto dentro de la escena.

Y la transformación en este GO primitivo señala la posición desde la que se comienzan a ver los objetos en la escena.

Todos los componentes internamente están creados mediante Scripts y utilizan atributos públicos para modificar sus valores, pero los Scripts también son un componente por sí mismos en cualquier GO. Se pueden crear y personalizar para asociarlos a un GO concreto que se encuentre en una escena.
Al asociar un Script a un GO, el script obliga al objeto a comportarse de una forma determinada.

PREFABS

Un GO es una entidad única en escena, pero a menudo, en los videojuegos necesitamos crear un mismo objeto muchísimas veces y no podemos crear un objeto con sus componentes desde cada vez que necesitemos usarlo. Es en estos casos cuando un GO se debe convertir en un GO prefab (prefabricado).
Un prefab es un Game Object preparado con los componentes necesarios y guardado para ser reutilizado en el proyecto.

Con ellos podemos usar un objeto con una combinación de componentes determinada, infinidad de veces y podemos crearlos mediante scripting o desde el panel de Hierarchy.

Para convertir un GO en prefab solo debemos crear una carpeta Prefab en el panel de Folders y arrastrar un objeto desde el panel de Hierarchy a dicha carpeta.

5. Scripts

Los Scripts desde el punto de vista de un desarrollador de Unity son el elemento más importante del motor gráfico, ya que es el componente que dota de vida a un Game Object.

Para que los scripts sean interpretados por el Motor Gráfico debemos utilizar la API de Unity. Para usar la API de Unity debemos importar las librerías al comienzo de un script de Unity.

La API de Unity es muy profunda por ello señalaremos algunas de sus propiedades más importantes:

- **Vector3.** Es la propiedad que se usa para representar dos cosas: puntos y direcciones. Se pueden usar para representar las posiciones de un objeto en un espacio, magnitudes, direcciones y movimiento.

- **Transform.** Es la propiedad que se usa para representar la posición, la escala y la rotación. La posición de un objeto está formada por propiedades Vector3 y la escala y la rotación dependen de la clase Component y permite que desde ella se acceda a la clase GameObject.

- **GameObject.** Es la propiedad a la que se accede para referenciar un objeto mediante scripting. Y permiten acceder desde ellos a los componentes que lo forman.

CLASE MONOBEHAVIOUR

Pero todas estas clases no significan nada sin la clase MonoBehaviour. Esta es la clase de la que debe derivar un script para poder ser utilizado en Unity. Desde esta clase podemos referenciar transformaciones, crear vectores y game objects.

Monobehaviour utiliza una serie de fases específicas con eventos determinados en cada fase:

Initialization Phase (Fase de Inicialización)

En esta fase se utilizan los métodos **Awake ()**, **OnEnable ()** y **Start ()**. Estos eventos sirven para ejecutar variables funciones en determinados momentos:

- **Awake.** Cuando el script es llamado y todos los objetos se inicializan. Esto permite acceder a GameObjects sin preocuparnos de si un objeto está instanciado o no.

- **OnEnable.** Cuando el gameobject que lleva el script es activado por primera vez o tras una desactivación. Este método es llamado intrínsecamente por los GOs siempre una vez.

- **Start.** Cuando se activa el script y antes del método Update. Es el evento más usado en los scripts y viene por defecto al crear un script en Unity. Esto sucede en el primer frame por segundo al ejecutar la aplicación.

Game Logic (Lógica de Juego)

Esta fase está relacionada con la lógica del juego. Los eventos son Update y LateUpdate.

- **Update.** Este evento se utiliza una vez por cada frame por segundo, es decir que se va repitiendo con el tiempo. Es el otro método que viene en los scripts Unity por defecto.

- **LateUpdate.** Es el evento que se utiliza tras realizar los cálculos de Update. Es útil si haces cálculos que dependen de los resultados de la función Update.

Input Events Phase (Fase de Eventos de Input)

Estos son todos los eventos relacionados con la pulsación de periféricos como joysticks, teclados y ratones. Sus eventos son por ejemplo **OnMouse[acción]** y **OnKey[acción]**.

Pausa o desahabilitación.

Estos son los eventos relacionados con la pausa, la desactivación de GameObjects como **OnDisable ()** que desactiva un GO o **OnAplicationQuit ()** que se activa cuando la aplicación se va a cerrar o **OnDestroy ()** que se ejecuta cuando un objeto se va a destruir.

Unity no obstante contiene muchísimos eventos, estos son solo una muestra de los más importantes.

Aquí muestro un ejemplo de un script de Unity. Esta es la clase robot de mi proyecto 3D. En ella usamos la API de Unity en Unity Engine y las Colecciones Genéricas de Unity.

1. Esta clase hereda de Monobehaviour para poder ser usada en un GO del motor gráfico. Al arrancar la app en el primer frame por segundo, desde el Game Object accedemos a componentes del GO y elementos que se encuentran en él (en este caso es una etiqueta que identifica al GO llamada tag).

2. Después en cada frame por segundo observamos si el robot está muerto o no y con transform accedemos a una función que hace que el robot mire en la dirección del jugador y con un componente de Inteligencia Artificial movemos su localización hacia el jugador.

3. Y finalmente si el robot está a una distancia calculada mediante la propiedad Vector3 ejecuta una función que dispara al jugador.

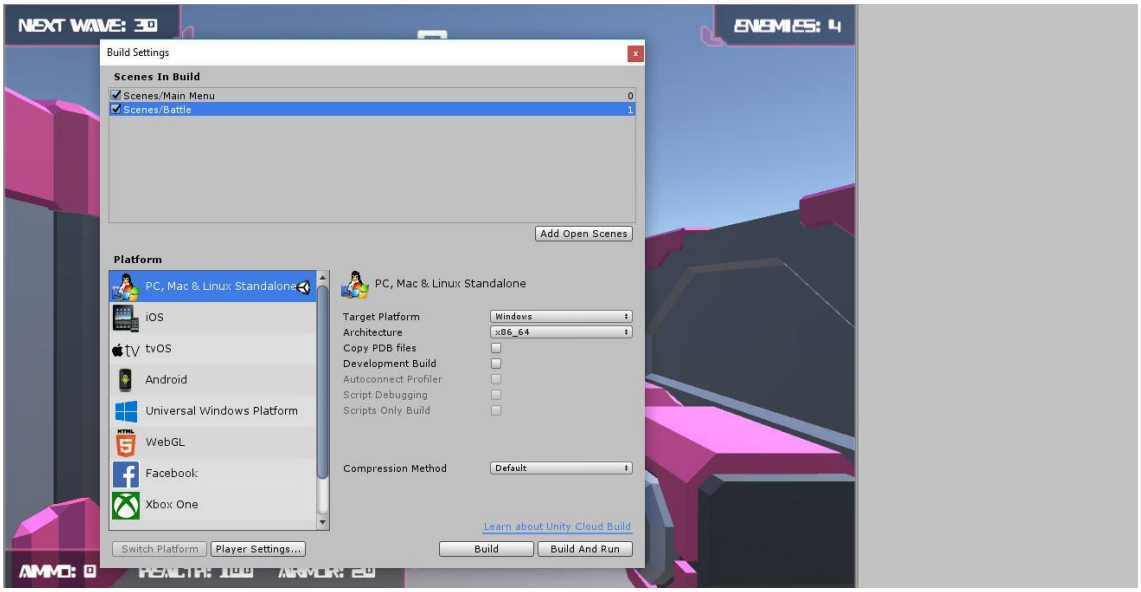
Robot.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class Robot : MonoBehaviour {

    [SerializeField] private string robotType;

    public int health;
    public int range;
    public float fireRate;
    public Transform missileFireSpot;
    public Animator robot;

    NavMeshAgent agent;
    Transform player;
    float timeLastFired;
    bool isDead;

    [SerializeField] GameObject missilePrefab;
    [SerializeField] AudioClip deathSound;
    [SerializeField] AudioClip fireSound;
    [SerializeField] AudioClip weakHitSound;


    void Start () {
        isDead = false;
        agent = GetComponent<NavMeshAgent>();
        player = GameObject.FindGameObjectWithTag("Player").transform;
    }


    void Update () {
        if(isDead){
            return;
        }

        transform.LookAt(player);
        agent.SetDestination(player.position);

        if(Vector3.Distance(transform.position, player.position) < range && Time.time - timeLastFired > fireRate){
                timeLastFired = Time.time;
                Fire();
        }

    }
```

6. Despliegue de la Aplicación

El despliegue en Unity es muy sencillo. Cuando tenemos una aplicación terminada en Unity debemos elegir la plataforma en la que vamos a utilizar dicha aplicación.
Unity nos muestra en el menú FILE y en el apartado BUILD SETTINGS todas las plataformas que podemos elegir.

Una vez elegida la plataforma, debemos asegurarnos de tener todas las escenas de la app en nuestro panel de builds. Si una escena falta, cuando queramos saltar a dicha escena sea un menú por el que se accede mediante un botón o un nivel del juego que aparece al pasarnos el nivel anterior, nos dará un error y la app no nos dirigirá a la escena a la que queremos. Cada escena se asocia con un número, este número es el orden en el que las escenas van a aparecer, normalmente solo nos interesa el numero 0 que es la primera y debemos colocar en dicha posición la escena correspondiente (normalmente es un menú principal). Y desde esa escena podremos ir a cualquiera sin importar el orden. Es decir, de la escena 0 podemos pasar a la escena 4 mediante un botón después regresar a la 3 y luego ir a la 1 etc.



Para las builds en aplicación Web seleccionamos HTML 5 que tiene asociada la API de WebGL si le damos al botón build nos dará como resultado una carpeta con un archivo HTML que tiene insertado únicamente nuestro juego. Si antes de darle al botón de Build le damos al botón de Player Settings… podremos modificar las características de WebGL como la cantidad de memoria que usa el navegador al ejecutar el juego, el tamaño del juego en el documento HTML etc.
El archivo HTML se puede modificar en cualquier editor de texto y después solo tenemos que subirlo a un servidor y nuestro juego estará Online y listo para jugarlo desde cualquier navegador.

7. Bibliografía

Documentación de Unity – Unity User Manual V. 2017.4
https://docs.unity3d.com/Manual/index.html
Gametopia — Curso Online Programación Unity – Nivel Medio por Diego Adrada (2018)