PROGRAMACIÓN MULTIMEDIA Y DISPOSITIVOS MÓVILES

UT5. Desarrollo de juegos 2D y 3D

Departamento de Informática y Comunicaciones

CFGS Desarrollo de Aplicaciones Multiplataforma

Segundo Curso

IES Virrey Morcillo

Villarrobledo (Albacete)

Índice

1. Módulo 1. Introducción al entorno

1.1. Motores de juego

1.1.1. ¿Qué es un motor de juegos?

Muchas plataformas implican muchos entornos de desarrollo. Aprender todos los entornos de desarrollo nativos de cada dispositivo es una tarea casi imposible y solo grandes compañías pueden tener equipos de desarrollo suficientemente grandes para permitirse esta aproximación.

Los motores de juegos son entornos cuyo objetivo no es otro que proporcionar a los desarrolladores herramientas más cómodas para desarrollar sus juegos y, en su mayoría, permitir generar aplicaciones y juegos para diversas plataformas, reduciendo de esta forma el coste de desarrollo.

1.1.2. Introducción a los motores de juegos

Ver video

1.2. Instalando Unity

Unity se encuentra en estos momentos disponible para Mac y Windows. Para su instalación, simplemente hemos de ir a su página web principal: http://unity3d.com

En ella tenemos que ir al apartado Get Unity donde podremos elegir entre las diferentes versiones disponibles de Unity.

Todas las versiones son equivalentes en cuanto a herramientas y potencia del editor. Pero la versiones no personales disponen de algunos complementos extra muy recomendables (Unity Analytics Pro, Team License, acceso a versiones beta, Splash Screen personalizable etc.)

La versión Personal Edition es perfecta para desarrollo Indie y para el perfecto seguimiento de este curso. Por tanto, para instalar Unity tenemos que ir al Free Download de la version Personal Edition. Una vez descargada tenemos que instalarla y registrarnos con nuestro email.

1.3. El editor de Unity y las vistas

1.3.1. Unity y las vistas

Ver video

1.3.2. Las vistas más importantes

Ver video

1.3.3. Unity no tiene herramientas de modelado

Una de las primeras preguntas que formula un iniciado en Unity es cómo puede modelar objetos y personajes en el editor. La respuesta no puede ser más rotunda: Unity no tiene herramientas de modelado.

El editor de Unity está concebido para:

- ✓ Importar elementos modelados en entornos específicos, como Blender, 3D Max, Maya, etc. Incluso, por supuesto, no sólo los modelos sino las animaciones.
- ✓ Distribuir estos elementos en una escena.

El modelado 3D es sumamente complejo y sin duda exige de herramientas específicas. No es el objetivo de Unity intentar emular estas herramientas tan poderosas.

Aunque Unity no dispone de herramientas de modelado, terceros han diseñado complementos que permiten al editor de Unity llevar a cabo estas tareas, pensando básicamente en el modelado de niveles y objetos sencillos. Son complementos de terceras compañías y generalmente con un coste adicional.

¿Cómo podemos utilizar algunos modelos sencillos para nuestros ejemplos?.

Tenemos claro que Unity no es una herramienta de modelado, pero ¿podemos utilizar algún modelo sencillo para nuestros ejemplos sin que tengamos que importar nada?

Efectivamente. Podemos ir al menú GameObject -> 3D Object y allí encontraremos, entre otros, elementos de geometría sencilla como Cubo, Esfera, Cápsula, Cilindro, Plano y Quad.

Abre Unity y pon un cubo (Cube) en la escena a partir del menú anterior. Lo utilizaremos en los vídeos siguientes.

¿Qué formato está recomendado cuando queramos importar a Unity?.

Aunque Unity admite muchos formatos, el .fbx es un formato recomendado en la importación de modelos y animaciones en el editor. Exporta desde programas de modelado 3D en este formato siempre que sea posible.

1.3.4. Vista escena - Elementos básicos

Ver video

1.3.5. Vista escena - Elementos adicionales

Ver video

1.3.6. Vista escena - Resumen de sus posibilidades

Herramientas de manipulación de objetos (teclas q,w,r,t)

- ✓ La primera es la '**Hand tool**'.
- ✓ Las otras 3 herramientas permiten la manipulación de objetos: **mover** (tecla 'w'), **rotar** ('e') y **escalar** ('r').

Para su funcionamiento hay que seleccionar un objeto y con ello aparecerá el correspondiente 'Gizmo' de la herramienta.

El Gizmo de la herramienta nos permitirá controlar el cambio de posición, escalado o rotación del objeto de forma global o en algún eje particular.

Las herramientas de manipulación actúan sobre un componente del GameObject denominado Transform y que almacena la posición, escala y rotación del objeto.

Ejercicio propuesto: crear un GameObject (esfera, cubo ...) y modificar su posición, escala y rotación, familiarizándose con sus posibilidades y observando en la vista 'Inspector' los valores de la componente 'Transform'. Ver lo que ocurre cuando se hace click en el centro del Gizmo.

Experimentar el efecto de pulsar la barra de espacio + Mayúsculas, y la selección de un objeto y la tecla 'F' (click-doble sobre el objeto en Hierarchy).

Herramientas de control de posición

- ✓ **Pivot / Center** permiten que los cambios se hagan con respecto al pivot del objeto o su centro. Son también especialmente interesantes cuando tenemos varios objetos seleccionados. 'Pivot' hará que los cambios se hagan respecto al 'pivot' de uno de ellos, 'Center' será respecto al centro de todos los seleccionados
- ✓ **Local / Global** serán especialmente útiles cuando apliquemos rotación a algún objeto. 'Local' hará que futuros cambios tengan en cuenta esta rotación previa (la transformación se hará respecto al sistema de coordenadas local del objeto), mientras que 'Global' hará

que las transformaciones se lleven a cabo siempre respecto al sistema de coordenadas global de la escena.

- ✓ La selección múltiple con Shift

Ejercicio propuesto: Crear varios GameObject, seleccionarlos y experimentar las diferencias de Pivot / Center. Bajo el mismo ejemplo, aplicar rotación a algún ejemplo y cambiar su posición experimentando las diferencias de Local / Global.

Herramienta 'Hand tool' (tecla 'q')

La navegación por la escena puede cambiar en base al sistema operativo y al ratón disponible (2 ó 3 botones)

La herramienta 'hand tool' cambiará en base a las distintas posibilidades: drag (mano), orbital (ojo), lupa (zoom):

- Ctrl (Windows) / Cmd (Mac) -> modo zoom
- Alt -> modo orbital
- La rueda del ratón puede usarse también para el zoom (Alt + botón derecho)
- Flechas del cursor -> modo 'walking' o caminante
- Botón derecho + WASD + QE -> Modelo 'flythrough'
- NO ES GENERALMENTE necesario seleccionar la herramienta 'hand tool' para interactuar con la escena, ya que estas teclas siguen activas en los otros modos.

Ejercicio: Crear varios objetos, manipularlos y navegar por la escena utilizando todas las posibilidades de la herramienta 'hand tool'

Snapping

Snapping (Edit -> Snap Settings)

Permite establecer unos valores de cambio para traslaciones, rotaciones y escalados.

Para que el objeto se someta a un cambio con 'snap', hay que pulsar la tecla Ctrl(Windows)/Cmd(Mac)

En Snap Settings se pueden cambiar las propiedades y también forzar a un objeto en concreto seleccionado a que se ajuste a los incrementos de snap.

Vertex Snapping

Con 'translate' y pulsando la tecla 'V' podemos seleccionar un vértice y ajustarlo a otro vértice de otro objeto. Esta opción es especialmente útil en la creación de niveles cuando queremos hacer encajar a la perfección diferentes secciones (sin solapamientos ni huecos)

En general, y más concretamente en la creación de niveles, las herramientas de Snapping y Vertex Snapping son imprescindibles.

1.3.7. Vista escena - Información adicional

La vista escena permite otras opciones de carácter más avanzado. Estas opciones están en la vista escena justo debajo de su solapa. Estas opciones no se precisan cuando iniciamos nuestro aprendizaje en Unity, pero pasamos a describir brevemente sus posibilidades.

El primer menú nos permite establecer el modo de visualización de los elementos de la escena (no sobre el juego, sólo en esta vista). Las más importantes son los modos 'shaded' (con textura), Wireframe (modo alámbrico) y ShadedWireFrame (mixto).

Los otros son de carácter avanzado, Render Paths (tipo de renderizado de los objetos, verde para deferred lighting, amarillo para forward rendering y rojo para vertex lit), el canal Alpha, las áreas de mayor carga gráfica (Overdraw) y Mipmaps que nos puede orientar sobre el adecuado tamaño de las texturas de los objetos (rojo indica textura más grande de los necesario, azul que la textura debería ser mayor).

El siguiente botón nos permite ver la escena en modo 2D o 3D.

El icono siguiente nos permite visualizar la escena con las luces realmente asignadas o con una luz ambiental por defecto (y que no se aplicaría en el juego final).

El botón de audio nos permite o no reproducir los sonidos en la vista de escena.

El siguiente botón nos permitirá ver/ocultar en la vista escena elementos como SkyBox, niebla, efectos de brillo de la cámara y materiales animados.

Gizmos nos permite establecer propiedades de los mismos.

Finalmente una zona de búsqueda para identificar elementos de la escena.

1.3.8. Vista jerarquía

Ver video

1.3.9. Vista juego

El objetivo es mostrar lo que verás en el juego a partir de sus cámaras.

Deberemos proporcionar al menos una cámara para que la escena pueda visualizar elementos.

Con estos botones arrancaremos el juego, lo pausaremos o lo ejecutaremos paso a paso.

ES IMPORTANTE tener en cuenta que **Unity** nos permite realizar modificaciones a los elementos de la escena mientras estamos ejecutando el juego, PERO estos cambios se desharán cuando el juego se detenga (es un comportamiento útil durante el desarrollo pero peligroso muchas veces).

En las opciones de la vista de escena podemos cambiar el aspect/ratio y resolución de la misma, activar que se maximice cada vez que ejecutamos y mostrar estadísticas respecto a los recursos consumidos en su ejecución.

Los cambios de relación de aspecto pueden ser muy interesantes para evaluar el comportamiento de nuestro juego en diferentes dispositivos.

1.4. Ejercicios propuestos

Un muy buen ejercicio para practicar lo aprendido consiste en modelar un cuerpo humano. Para ello hay que utilizar los elementos ya predefinidos de los que dispone Unity, como Esferas, cubos, cilindros y cápsulas. A base de estos elementos y mediante su cambio de posición, escala y rotación, podemos conseguir modelar las diferentes partes del cuerpo. Por ejemplo, una esfera para la cabeza, un cubo para el torso, cilindros para antebrazo y brazo, etc.

Es muy conveniente utilizar también relaciones padre/hijo. De esta forma, la mano debería ser hija del brazo y este del antebrazo... y así sucesivamente. Por tanto, si movemos o rotamos el antebrazo, también lo harán los hijos/nietos etc. Es una muy habitual forma no sólo de modelar elementos articulados sino de agruparlos jerárquicamente para facilitar su posterior animación.

Al hacer este ejemplo, es importante que no se cambie la escala de los elementos intermedios, es decir, podemos mover o rotarlos, pero hay que evitar escalarlos (provocaría efectos no deseables).

2. Módulo 2. Conceptos básicos de Unity

2.1. Conceptos básicos de Unity

2.1.1. Conceptos básicos

Ver video

2.2. Nuestro primer ejemplo – arquitectura de un proyecto

2.2.1. Creando un primer ejemplo

Ver video

2.2.2. Posicionando elementos

Ver video

2.2.3. Creando materiales

Ver video

2.3. Nuestro primer ejemplo – script

2.3.1. Creando el script (I)

Ver video

2.3.2. Creando el script (II)

Ver video

2.3.3. Creando el script (III)

Ver video

2.3.4. Time.deltaTime

Ver video

2.3.5. Creando el script (IV)

Ver video

2.3.6. Creando el script (V)

Ver video

2.4. Ejercicios propuestos

Puede resultar interesante modificar los materiales para que podamos hacer algo más que simplemente cambiar el color del objeto. Podemos buscar en internet texturas (imágenes) que queramos aplicar como 'piel' a nuestros elementos de la escena.

Para ello simplemente tenemos que importar en Unity estas texturas (arrastrándolas simplemente desde una carpeta hasta el propio editor de Unity, concretamente sobre la vista 'Project'). Es aconsejable como siempre crear una carpeta específica en 'Project' donde poner estas texturas.

Para cambiar uno de los materiales de los objetos de la escena, lo único que tenemos que hacer es volver a dejar el color del 'Albedo' del material a blanco (para que el color seleccionado no 'tiña' nuestra textura). En el mismo campo Albedo podemos aplicar esta nueva textura que hemos importado, con arrastrar y soltar sobre el cuadrado a la izquierda de 'Albedo' o seleccionándolo pulsando sobre el círculo también a la derecha del 'Albedo'.

El resultado ha de ser que nuestra esfera, cubos o plano de la escena deben ahora estar 'vestidos' con la textura importada.

3. Módulo 3. Introducción al motor de físicas

3.1. Introducción a la física

3.1.1. Más sobre GameObjects y Components

Ver video

3.1.2. La física en Unity

Ver video

3.2. Primer ejemplo físico

3.2.1. De cinemático a físico

Un pequeño truco...

Cuando estemos trabajando con MonoDevelop como editor de nuestros scripts, el editor nos va a sugerir nombres completos de métodos, variables etc. de forma que podemos evitar teclearlos completamente. Este sistema inteligente a veces no hace bien su trabajo y nos impide teclear lo que realmente queremos.

Un ejemplo lo tendremos cuando queramos teclear la variable en minúsculas 'rigidbody' y el editor se empeñe en sustituirlo por 'Rigidbody'. Ambos por supuesto no son equivalentes.

Cuando suceda que no nos interesa que este sistema inteligente del editor funcione, basta que empecemos a teclear lo que realmente queremos y si en la lista de sugerencia no está lo que queremos teclear, pulsaremos la tecla 'ESC' para evitar que el editor se empeñe en sustituir lo que queremos por su sugerencia.

La tecla 'ESC', por tanto, es a menudo útil cuando tecleamos nuestros scripts en MonoDevelop.

Ver video

3.2.2. Saltando

Ver video

3.2.3. Los Prefabs

Ver video

3.2.4. Obstáculos y Prefabs

Ver video

3.2.5. Añadiendo un sonido

Ver video

3.3. Ejercicios propuestos

Probar las consecuencias de utilizar diferentes valores de 'Drag' y 'Angular drag' de nuestra esfera.

Probar diferentes valores para la fuerza de salto de nuestra esfera.

Probar diferentes valores de masa para nuestra esfera y combinarlo con diferentes fuerzas y ver las consecuencias al colisionar con una pila de cubos.

4. Módulo 4. Unity y las diferentes plataformas

4.1. Unity y las diferentes plataformas

4.1.1. Unity y las diferentes plataformas

Ver video

4.2. Consideraciones de pantalla

4.2.1. Relación de aspecto y otras consideraciones

Ver video

4.2.2. Relación de aspecto y la vista Game

Ver video

4.3. La compilación condicional

4.3.1. La compilación condicional

Ver video

4.4. Unity en desktop y consolas

4.4.1. Unity en desktop y consolas

Ver video

4.5. Unity y los dispositivos móviles

4.5.1. Unity y los dispositivos móviles

Ver video

4.5.2. Configurando Android

Si disponemos de un dispositivo Android, podemos con la versión **Personal Edition** de **Unity** desplegar nuestra aplicación sobre él. Para ello tenemos que seguir una serie de pasos. Los vamos a ver a continuación.

Toda esta información puede ser consultada en la documentación original de Unity en http://developer.android.com/sdk/index.html.

Descargamos Android SDK (no es necesario descargar el Android Studio, en '*Other download Options*' podemos seleccionar *SDK Tools only*).

Instalamos el SDK. Tenemos que instalar al menos una plataforma Android con API level igual o mayor que 9 (Platform 2.3 o superior), las Platform Tools, y los USB drivers si usamos Windows (http://developer.android.com/sdk/installing/index.html).

En Windows puede ser necesario instalar drivers propios del fabricante del dispositivo (en algunos modelos este paso por desgracia puede ser complejo).

Es importante también activar la opción **USB Debugging** (depuración USB) en el dispositivo, yendo al menú de nuestro dispositivo Android Ajustes->Opciones de desarrollo. Si no aparece esta opción en Ajustes, hay que ir a Ajustes->Información del teléfono->Número de compilación y pulsar esta opción varias veces.

Por último, hay que añadir en el editor de Unity (**Unity->Preferences**) dónde está instalado (en qué carpeta) el Android SDK.

Con **Build** generamos el .apk, con **Build & Run** podemos lanzar el juego directamente a ejecutar en el dispositivo.

Dependiendo del modelo de nuestro dispositivo Android podemos encontrarnos con más o menos dificultades en poder desplegar nuestro juego. Básicamente **Unity** precisa de los mismos pasos que son necesarios si queremos desarrollar aplicaciones Android con sus herramientas oficiales de desarrollo. En Internet podemos encontrar posiblemente todas las dudas de instalación de los diferentes modelos, no obstante aconsejamos el uso del foro para que el estudiante exponga aquí los problemas que haya podido encontrar por si otros estudiantes pueden facilitar una solución.

4.5.3. Configurando iOS

La configuración de los dispositivos iOS precisa disponer de un ordenador Mac con la herramienta Xcode instalada.

Necesitamos de una cuenta de desarrollador iOS para probar nuestro juego en un dispositivo (aunque parece que en futuras versiones de Xcode puede que no sea necesario para probar nuestra aplicación en un dispositivo concreto).

Toda la información la podemos encontrar en http://docs.unity3d.com/Manual/iphone-GettingStarted.html.

De nuevo, podéis usar el foro para resolver dudas en el despliegue de aplicaciones en iOS.

4.5.4. Creando el proyecto Mobile

Vamos a intentar modificar nuestro ejemplo de la esfera para poder usarlo en un dispositivo móvil. Para ello, en primer lugar, vamos a familiarizarnos con algunas de las posibilidades de los dispositivos móviles y cómo gestionarlas en **Unity**. Para ello vamos a crear un nuevo proyecto al que llamaremos **Mobile**.

Creamos un nuevo proyecto, al que llamaremos **Mobile.**

Creamos dos carpetas en el proyecto: Scripts y Scenes.

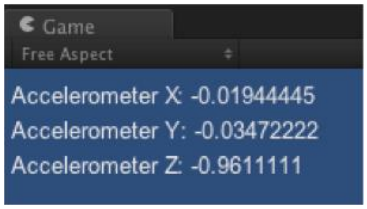
Guardamos nuestra escena **Untitled** como **Mobile** en el directorio Assets->Scenes.

Creamos un nuevo script C# en la carpeta **Scripts** al que llamamos **MobileInput.**

```
using UnityEngine;
using System.Collections;
public class MobileInput : MonoBehaviour {
	void OnGUI() {
		GUI.Label(new Rect(5, 5, 200, 20), "Accelerometer X: " + Input.acceleration.x);
		GUI.Label(new Rect(5, 25, 200, 20), "Accelerometer Y:" + Input.acceleration.y);
		GUI.Label(new Rect(5, 45, 200, 20), "Accelerometer Z:" + Input.acceleration.z);
	}
}
```

Asignamos este script al GameObject Camara de la escena.

Ejecutamos asegurándonos que tenemos lanzado el **Unity Remote** en el dispositivo y observamos los cambios producidos al mover el dispositivo (o directamente lo ejecutamos sobre el dispositivo si lo queremos hacer directamente).



Con el movimiento del dispositivo móvil, su acelerómetro reporta cambios de aceleración lineal a través de sus 3 ejes primarios en el espacio 3D.

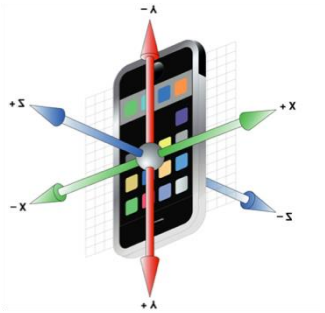
Podemos usar esta información para detectar cuál es la orientación del dispositivo y cualquier cambio producido sobre ésta.

La aceleración a lo largo de un eje se devuelve en unidades G (1G = 9.81 m/s^2).

Un valor de 1.0 representa una carga de 1 gravedad en ese eje.

A partir de la versión 4.0, los ejes de los acelerómetros están alineados con la vista de juego (Game View), con lo que hay que fijarse en la orientación que definimos en 'Player Settings' (landscape, portrait etc.) a la hora de saber cómo se orientan la X, Y, Z.

Si hemos seleccionado 'portrait', sosteniendo el dispositivo en su posición vertical original (con la pantalla hacia el usuario), el eje X es positivo hacia la derecha, el eje Y es positivo directamente hacia arriba y el eje Z es positivo hacia el usuario.



Usando Input.acceleration podemos llevar a cabo la lectura de los 3 ejes.

También podemos usar **Input.deviceOrientation** para que nos devuelva una valoración discreta de la orientación del dispositivo, en concreto una enumerado del tipo **DeviceOrientation**: **FaceDown**, **FaceUp**, **LandscapeLeft**, **LandscapeRight**, **Portrait**, **PortraitUpsideDown**, **Unknown.**

4.5.5. Versión móvil con acelerómetros

Ver video

4.5.6. Ampliando el proyecto Mobile

Retomemos el proyecto **Mobile**, y añadamos el siguiente código:

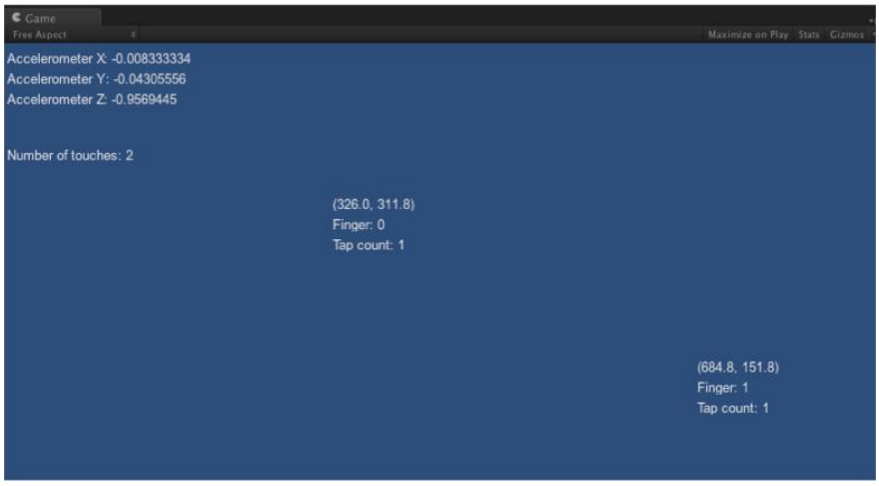
```
using UnityEngine;
using System.Collections;
public class MobileInput : MonoBehaviour {
        void OnGUI() {
                GUI.Label(new Rect(5, 5, 200, 20),  "Accelerometer X: " + Input.acceleration.x);
                GUI.Label(new Rect(5, 25, 200, 20), "Accelerometer Y: " + Input.acceleration.y);
                GUI.Label(new Rect(5, 45, 200, 20), "Accelerometer Z: " + Input.acceleration.z);
                GUI.Label(new Rect(5, 100, 200, 20), "Number of touches: " +
Input.touchCount);
```

```
            for (int i = 0; i < Input.touchCount; i++) {
                PrintTouchData(Input.touches[i]);
            }
        }
        void PrintTouchData(Touch t) {
            GUI.Label (new Rect(t.position.x, Screen.height - t.position.y, 100, 20), t.position.ToString());
            GUI.Label (new Rect(t.position.x, Screen.height - t.position.y + 20, 100, 20), "Finger: " + t.fingerId.ToString ());
            GUI.Label (new Rect(t.position.x, Screen.height - t.position.y + 40, 100, 20), "Tap count: " + t.tapCount.ToString());
        }
    }
```

Hemos completado el script del proyecto **Mobile** para que, además de la información de los acelerómetros, podamos visualizar cuál es el resultado de tocar la pantalla del dispositivo con uno o más dedos.

Cuando el jugador toca la pantalla **Input.touchCount** nos va a devolver el número de dedos (toques) que están sobre la pantalla. En el array **Input.touches** vamos a tener cada uno de esos toques del tipo **Touch**. Lo que hacemos es mostrar en un bucle la información de cada uno de los toques en pantalla.

La información que imprimimos de cada toque es su posición en coordenadas x,y de pantalla (**t.position**), el identificador de dedo (**t.fingerId**) y el número de 'taps' de ese dedo. Un 'tap' no es más que la acción de tocar y dejar de tocar la pantalla de forma rápida, de forma que **Unity** nos permite detectar si estamos haciendo un toque, un doble toque etc. con objeto de poder asociar diferentes acciones a estos toques.



Otra propiedad de **Touch** muy interesante es la fase (**.phase**), que devuelve un enumerado del tipo **TouchPhase**, cuyos valores más interesantes son: **Began** (cuando tocamos por primera vez), **Moved** (cuando movemos el dedo), **Stationary** (permanecemos tocando pero no movemos el dedo) y **Ended** (hemos dejado de tocar la pantalla con ese dedo).

La fase pasará primero por **Began**, alternará entre **Stationary** y **Moved**, y finalizará en **Ended.**

4.5.7. Saltando con un toque

Ver video

4.6. Ejercicios propuestos

Para aquellos que tengan un dispositivo móvil y la posibilidad de utilizarlo con Unity, un interesante ejercicio es intentar que la esfera salte cuando apliquemos un movimiento brusco en el acelerómetro Z (si tenemos la pantalla del móvil mirando hacia arriba, hacer un movimiento brusco hacia arriba, el acelerómetro Z, con objeto de simular lo que ocurriría en la realidad si la esfera estuviera sobre el plano que forma la pantalla del dispositivo).

Basándose en el código que ya tenemos para saltar, tendríamos que cambiar el if. En lugar de saltar cuando hacemos un toque, deberíamos hacerlo cuando el valor del acelerómetro Z superase un determinado valor positivo. Llevar a cabo diferentes pruebas para determinar un valor adecuado.

5. Módulo 5. Scripting

5.1. C# y Unity

5.1.1. Aprendiendo C#

El lenguaje de programación que estamos utilizando en el curso es C#. Unity también permite el uso de Javascript, pero es C# el lenguaje preferido en estos momentos por los desarrolladores por su potencia y cualidades.

C# es un lenguaje estándar fácil de aprender si se tienen conocimientos previos de Java o C++.

En este curso de introducción no es necesario llegar a dominar este lenguaje de programación pero sí que es un elemento fundamental para aquellos que quieran convertirse en expertos en Unity, dado que el dominio del lenguaje nos puede permitir hacer más cosas y mejores en nuestros scripts.

De nuevo, no es necesario conocer con profundidad C# para seguir el curso. Con unas nociones básicas de programación el alumno podrá ir siguiendo los ejemplos sin demasiadas dificultades. No obstante, como material únicamente complementario, los estudiantes más interesados en este lenguaje pueden conocer más al respecto de sus características en el siguiente pdf: C-Sharp_y_Unity.

Los alumnos interesados quedan invitados a leer esta información complementaria. Por descontado, existe en internet cantidad de cursos y tutoriales de C# y algunos incluso centrados en C# y Unity.

5.2. Elementos básicos de los scripts

5.2.1. Scripts y métodos básicos

Ver video

5.3. Colisiones y triggers

5.3.1. Detección de colisiones

Ver video

5.3.2. Ejemplo de colisiones

Ver video

5.3.3. Triggers

Ver video

5.3.4. Ejemplos de triggers

Ver video

5.4. Comunicación entre scripts

5.4.1. Acceso a componentes

Ver video

5.4.2. Comunicación entre objetos

Ver video

5.4.3. Ejemplo de comunicación

Ver video

5.5. Instalación de Prefabs

5.5.1. Creando objetos dinámicamente

Ver video

5.6. Corrutinas

5.6.1. Corrutinas

Ver video

5.6.2. Ejemplo de corrutina

Ver video

5.7. Ejercicios propuestos

Añadir a la esfera un nivel de vida (p.e. un 'int vida = 100;' como nuevo atributo). Cada vez que la esfera colisione con un cubo le restaremos una unidad a ese nivel de vida. Cuando el nivel de vida sea 0 deberemos imprimir en la consola "La esfera ha muerto".

Modificar la corrutina y añadir lo necesario para que, de forma aleatoria, no sólo se instancies cubos sino cápsulas. Las cápsulas también deben morir cuando pasen de un determinado valor de 'y'.

6. Módulo 6. Unity en acción

6.1. Un ejemplo completo

6.1.1. Un ejemplo completo

Vamos a abordar a continuación un ejemplo completo, en el que vamos a crear varias escenas, un juego con todo lo que hemos aprendido y que resulta sencillo poder ampliar y que termine siendo un juego del que estar orgullosos.

En la colección de vídeos que podemos ver a continuación, paso a paso, vamos a ir construyendo este juego. Primero como versión escritorio (windows, mac, linux) para finalmente aportar los cambios necesarios para también obtener una versión móvil del mismo preparado para pantallas táctiles.

Vamos a necesitar la descarga de dos ficheros que utilizaremos en el desarrollo del proyecto. Descarga estos dos ficheros y asegúrate de tenerlos a mano cuando sigas los vídeos.

6.1.2. DestroyStones -I- Carpetas y prefabs

Ver video

6.1.3. DestroyStones -II- Main loop

Ver video

6.1.4. DestroyStones -III- Time.timeScale

Ver video

6.1.5. DestroyStones -IV- Script Stone

Ver video

6.1.6. DestroyStones -V- OnMouseDown

Ver video

6.1.7. DestroyStones -VI- Prefab explosión

Ver video

Para no tener problemas de rendimiento en algunas plataformas, resulta interesante poder eliminar las instancias de las explosiones. Estas instancias no son automáticamente destruidas y como podemos ver si jugamos al juego, si nos fijamos en la vista jerarquía, a medida que vayamos destruyendo piedras, las instancias de las explosiones permanecen activas.

Un cambio sencillo para evitar esto consiste en ir al script Stone y modificar la primera línea que tenemos en el OnMouseDown, haciendo lo siguiente:

Destroy(Instantiate(explosion, transform.position, Quaternion.identity), 3.3f);

Es decir, programamos la autodestrucción de la instancia del objeto que creamos para que transcurridos 3.3 segundos (1.1 en realidad por el efecto del Time.timeScale) la instancia de la explosion sea destruida.

6.1.8. DestroyStones -VII- GameManager

Ver video

6.1.9. DestroyStones -VIII- UI

Ver video

6.1.10. DestroyStones -IX- Añadiendo escenas

Ver video

6.1.11. DestroyStones -X- OnClick

Ver video

En las recientes versiones de Unity, el método Application.LoadLevel("nombre de la escena") ha sido substituído. No obstante, aún seguirá siendo funcional un tiempo.

Podemos cambiarlo a lo que nos sugiere ahora Unity, para ello tenemos que hacer lo siguiente:

Añadir un nuevo 'using' en la parte superior de nuestro script:

using UnityEngine.SceneManagement;

Ahora en lugar de Application.LoadLevel("nombre de la escena") debemos utilizar SceneManager.LoadScene("nombre de la escena").

6.1.12. DestroyStones -XI- Mostrando resumen

Ver video

6.1.13. DestroyStones -XII- Condición fin del juego

Ver video

6.1.14. DestroyStones -XIII- Build Settings

Ver video

6.1.15. Haciendo móvil nuestro ejemplo

En principio no tenemos que hacer nada en especial en este ejemplo para que funcione en plataformas móviles, dado que OnMouseDown funciona en dispositivos táctiles. Si quisiéramos utilizar varios dedos simultáneamente para destruir piedras, convendría utilizar Input.touchCount (número de toques simultáneos) e iterar por los mismos (array Input.touches) para gestionarlos adecuadamente.

Los materiales que hemos creado para nuestras piedras están basados en el shader Standard. Este shader puede resultar muy lento en algunas plataformas móviles. Una solución es utilizar un shader específico para móviles. Podemos seleccionar el material y en el Inspector, en la parte superior, nos permite seleccionar un Shader distinto al Standard, por ejemplo podemos ir a la sección Legacy Shader y seleccionar el Difusse.

6.2. Juegos First and third person

6.2.1. Juegos First and Third Person

Los juegos en primera y tercera persona (conocidos como First Person, Third Person) son muy comunes y es interesante poder abordarlos en Unity.

Afortunadamente Unity, gracias a un conjunto de recursos (Standard Assets) disponibles tras su instalación, nos facilita enormemente el desarrollo de estos dos tipos de juegos.

Los primera persona son los típicos juegos donde la cámara se sitúa en los ojos del personaje y como mucho vemos sus manos o armas.

Los tercera persona nos permite ver completamente nuestro personaje, estando la cámara normalmente detrás del personaje.

En los siguientes vídeos podremos ver cómo podemos crear una plantilla inicial para estos dos tipos de juegos tan comunes.

6.2.2. Preparando el proyecto

Ver video

6.2.3. First person

Ver video

6.2.4. Third person

Ver video

6.3. Punto final

6.3.1. Punto final

Ver video

6.4. Ejercicios propuestos

Son muchas las ampliaciones que DestroyStones puede tener hasta convertirlo en un juego final. Algunas sugerencias:

- ✓ Añadir otros elementos a parte de las piedras, como elementos que NO debemos de destruir bajo penalización
- ✓ Añadir niveles, en los que la velocidad de los elementos y el nivel de exigencia aumente
- ✓ Añadir música de fondo al juego, sonidos de explosión etc.
- ✓ Pantallas inicial, final y de configuración como las de la mayoría de juegos
- ✓ Etc.