UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño y programación de videojuegos

UNLVIRTUAL

Manipulación de objetos en 3D

Unidad 2: Desarrollo de Video Juegos 3D optimizados para mobile en Unity

Primeros pasos

Docentes
Jaime Fili
Nicolás Kreiff

Contenidos

Desarrollo de VideoJuegos 3D optimizados para *mobile* en Unity 3

Primeros pasos: El primer proyecto en Unity 3

GameObjects 4

Componentes 5

La clase Component 6

La clase Behaviour 7

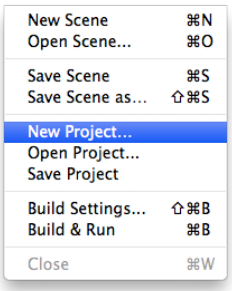
La clase MonoBehaviour 7

El primer componente 9

Desarrollo de VideoJuegos 3D optimizados para *mobile* en Unity

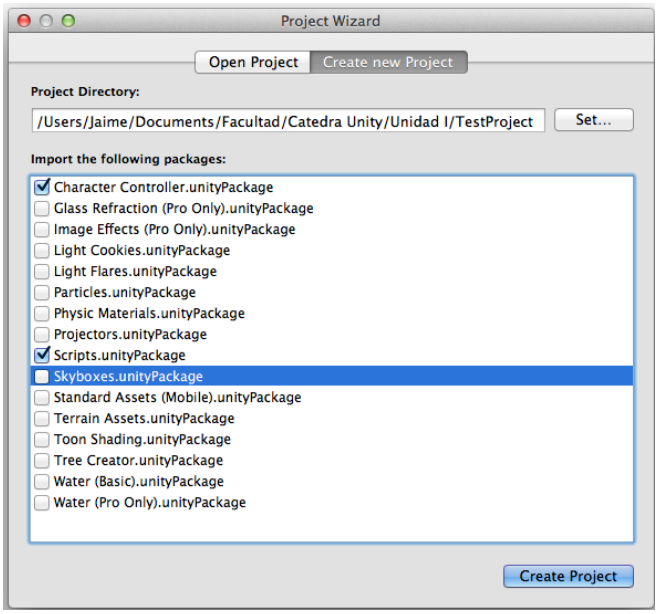
Primeros pasos: El primer proyecto en Unity

Luego de la introducción al mundo de Unity es hora de comenzar a trabajar en el primer proyecto. Para ello, se elegirá la opción *File* > *New Project…* como se indica en la imagen.

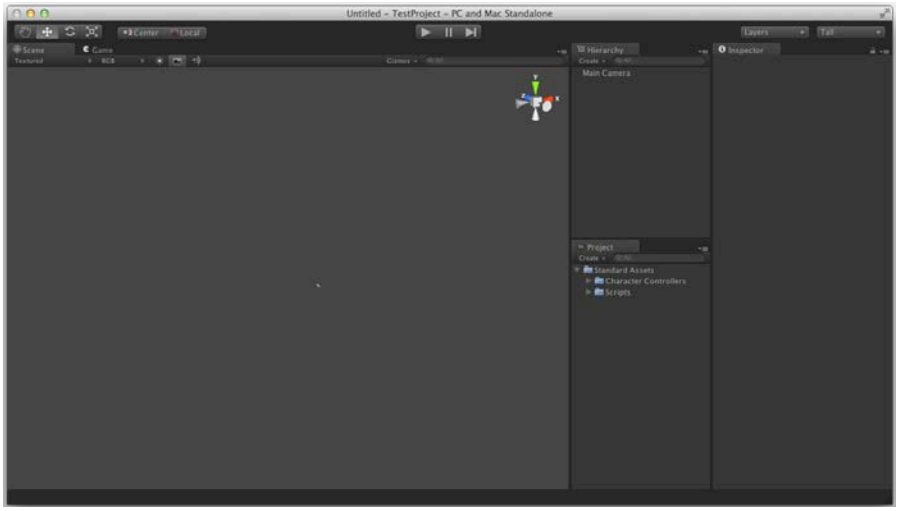


Al hacer esto aparecerá el *Project Wizard;* una simple pantalla que permitirá elegir la configuración inicial del proyecto. En esta pantalla se definirá dónde se desea guardar el proyecto pero lo más interesante es que ofrece también otras opciones, que se encuentran disponibles en el apartado *Import the following packages*. Por defecto, Unity trae un conjunto de librerías, componentes, shaders, para facilitar los desarrollos. Como se puede apreciar en la imagen a continuación, algunos paquetes están disponibles solamente para las licencias "PRO" de Unity.

El contenido de estos paquetes es amplio y muy variado; quedará a elección del usuario importar o no alguno de ellos en su proyecto. Si ningún paquete fuera seleccionado en el inicio y el usuario deseara agregarlo luego, podrá hacerlo desde el menú *Assets* > *Import Package*. También a través de esta opción se podrán agregar paquetes desarrollados por terceros o incluso trasladar partes de un proyecto a otro utilizando los "Unity Packages" (que se verán más adelante).



Una vez seleccionada la ubicación de destino del proyecto, y habiendo elegido (o no) paquetes a importar, se presionará la opción "Create Project". Acto seguido, Unity se cerrará y comenzará la creación del nuevo proyecto, incorporando todo lo que el usuario haya seleccionado.



Una vez finalizado el proceso, se apreciará que en la pestaña *Proyect* (si es que el usuario seleccionó alguno de los paquetes de Unity) se encontrará una carpeta denominada *Standard Assets*, que contendrá todo lo que el usuario haya incorporado a su proyecto. Existe también un paquete llamado *Standard Assets Mobile* que incluye versiones optimizadas de los recursos provistos por Unity para dispositivos móviles.

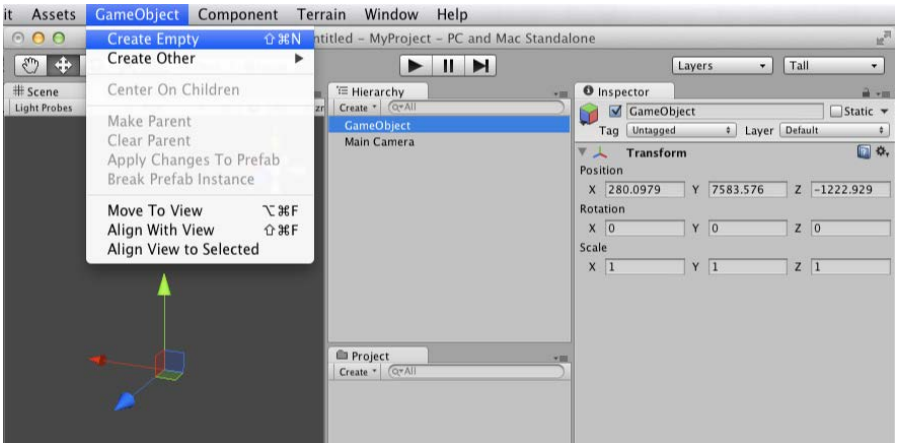
Ahora, con el proyecto limpio para comenzar a trabajar, se creará el primer *GameObject*.

GameObjects

De acuerdo a la documentación de Unity, *GameObject* es la clase base de todas las entidades en una escena de juego. En base a esta somera definición y desde la perspectiva del desarrollo uno podría pensar que debería derivar o heredar de esta clase para definir los distintos tipos de entidades que se necesitarán en el juego; sin embargo no es así.

En principio, un *GameObject* no es "algo" en sí mismo; es una entidad abstracta y carente de toda funcionalidad intrínseca. A esta entidad se le otorga un significado mediante la adición de *componentes*, que son los que definen sus características y comportamiento.

El editor de Unity permitirá crear un nuevo *GameObject* en la escena a través del menú homónimo. La primera opción se utilizará para crear un *GameObject* vacío, es decir, sin ningún *componente* que defina su función en la escena.



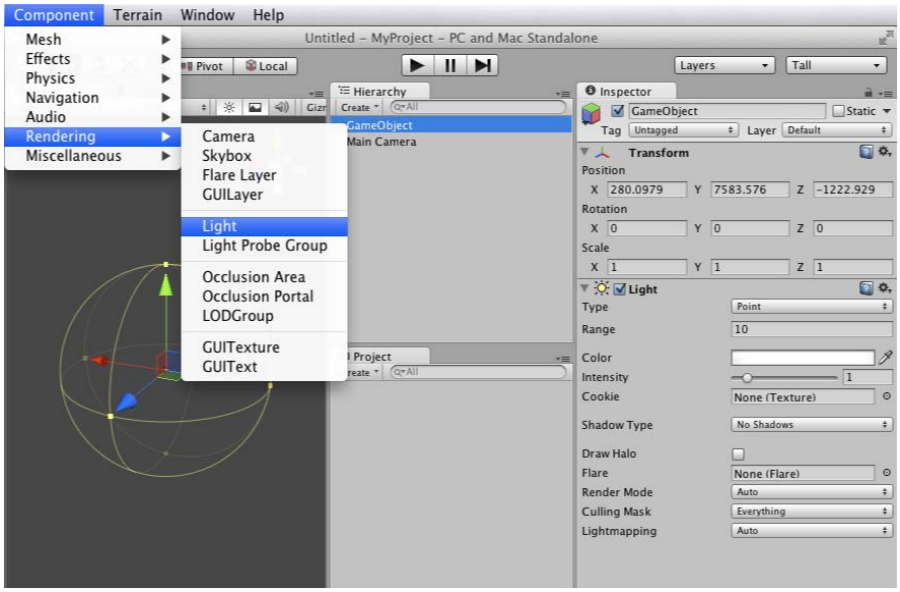
Como se puede observar en la imagen existe un componente que es imprescindible para cualquier *GameObject*, su *Transform*. Este componente lo sitúa en el espacio tridimensional de la escena y define el sistema de coordenadas para sus posibles hijos, ya que es también responsable de permitir el anidamiento.

Se dirá que un *GameObject* **A** está anidado, o que es hijo de un *GameObject* **B** cuando el componente *Transform* de **A** tenga como padre al componente *Transform* de **B**. Esto puede expresarse mediante el siguiente código:

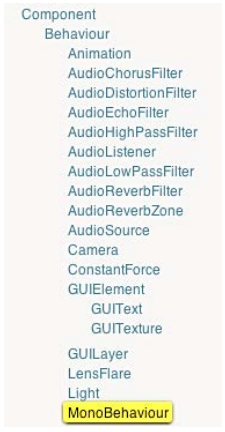
```
gameObjectA.transform.parent = gameObjectB.transform;
```

O directamente desde el *Editor* de Unity arrastrando el *GameObject* **A** sobre el *GameObject* **B**.

El submenú *Create Other* presenta una lista de elementos que ya poseen un significado claro y que son de uso muy frecuente para el desarrollo de cualquier juego. Si se añade a la escena una luz direccional, por ejemplo, podrá verse que la única diferencia que existe con un *GameObject* nuevo es el componente *Light*. Éste es el componente que lo define como una luz y contiene todas sus propiedades. A la vez, cualquier *GameObject* puede convertirse en una luz si se le agrega el componente *Light* (*Component* > *Rendering* > *Light*).



Componentes



Como se ha dicho, los *componentes* se adjuntan a un *GameObject* para otorgarles un significado. Unity no se esfuerza mucho más para definirlos: “Base class for everything attached to GameObjects” (ver link a la definición de la clase Component en la documentación de Unity a continuación).

Los componentes constituyen el punto de entrada para definir el comportamiento personalizado de los objetos del juego a crear.

Para definir un componente propio se deberá heredar de la clase *MonoBehaviour* que, como puede verse en el extracto de la jerarquía de clases de Unity, hereda a su vez de *Behaviour*; uno de los derivados directos de *Component*.

Antes de continuar es conveniente analizar lo que proporcionan estas clases a la hora de desarrollar un componente. Unity posee un *API* muy amplio; por lo cual estudiarlo un poco nunca está de más y evitará codificar innecesariamente cosas que el motor ya resuelve.

La clase Component

http://unity3d.com/support/documentation/ScriptReference/Component.html

Las siguientes son las variables de la clase:

Variable	Descripción
transform rigidbody camera light animation constantForce renderer audio guiText networkView guiTexture collider hingeJoint particleEmitter particleSystem	Las primeras 15 variables son accesos rápidos a componentes que uno comúnmente puede (pero no necesariamente debe) encontrar adjuntos a un *GameObject*. La de uso más frecuente es *transform*. Mediante esta variable se obtiene acceso al componente *Transform*, que permite manejar la ubicación en el espacio del objeto en la escena. Hay que recordar que, a excepción de *transform*, el resto de las variables son nulas (*null*) si no se instala un componente de ese tipo en el *GameObject*. Al intentar accederlas directamente se obtendrá una excepción por el acceso a un puntero nulo (*NullPointerException*). Como se verá más adelante, es posible obtener los componentes adjuntados a un objeto pero es mucho más eficiente accederlos a través de estas variables (siempre teniendo en cuenta que el componente exista en el objeto para no incurrir en errores en tiempo de ejecución).
gameObject	Mediante esta variable se accede al *GameObject* donde se encuentra instalado el componente. Su principal función es permitir el acceso a las funciones de la clase *GameObject* para instalar nuevos componentes en tiempo de ejecución.
tag	Es un acceso rápido al *tag* del *GameObject*. Se obtiene el mismo valor consultando *gameObject.tag*

Los métodos de esta clase son una extensión de los métodos de la *GameObject*, que es la verdadera encargada de manejar los componentes que contiene. De hecho, la clase *Component* solo posee métodos para consultar por otros componentes en el mismo objeto pero no para agregarlos. Como se mencionó antes, se pueden agregar nuevos componentes al *GameObject* a través de la variable que brinda acceso al mismo.

Los siguientes son los cuatro métodos más importantes de esta clase.

Método	Descripción
GetComponent	Retorna un componente dado su tipo. Existen tres variantes de este método: Light light1 = (Light) GetComponent(typeof(Light));Light light2 = (Light) GetComponent(“Light”); Light light3 = GetComponent<Light>(); Tanto *light1* como *light2* y *light3* harán referencia al

	mismo componente o bien serán nulos; en este caso no será necesario el *cast*. Esta última es la forma más adecuada y recomendada.
GetComponentInChildren	Retorna un componente dado su tipo, buscando tanto en el *GameObject* actual como en cualquiera de sus hijos, a la primera ocurrencia. Existen dos variantes de este método: Light light1 = (Light) GetComponentInChildren(typeof(Light)); Light light2 = GetComponentInChildren<Light>(); Tanto *light1* como *light2* harán referencia al mismo componente o bien serán nulos; en este caso no es necesario el *cast*. Esta última es la forma más adecuada y recomendada.
GetComponents	Idem *GetComponent* pero retorna todos los componentes del tipo especificado. Existen dos variantes del método: MyComp[] cmps1 = (MyComp[]) GetComponents(typeof(MyComp)); MyComp[] cmps2 = GetComponents<MyComp>(); En el último caso no es necesario el *cast* y esta es la forma más adecuada y recomendada.
GetComponentsInChildren	Idem *GetComponentInChildren* pero retorna *todos* los componentes del tipo especificado en el *GameObject* actual o en cualquiera de sus hijos. Existen dos variantes del método: MyComp[] cmps1 = (MyComp[]) GetComponentsInChildren(typeof(MyComp)); MyComp[] cmps2 = GetComponentsInChildren<MyComp>(); Nótese nuevamente que en el último caso no es necesario el *cast*. Esta última es la forma más adecuada y recomendada.

La clase Behaviour

http://unity3d.com/support/documentation/ScriptReference/Behaviour.html

El segundo punto en la recorrida por la herencia de la clase *MonoBehaviour* es su superclase, *Behaviour*. Aquí está su corta lista de variables:

Variable	Descripción
enabled	Este atributo booleano indica si el componente será actualizado o no. Se verá más adelante qué significa esto.

No hay ningún método al que hacer referencia aquí porque esta clase simplemente no los tiene.

La clase MonoBehaviour

http://unity3d.com/support/documentation/ScriptReference/MonoBehaviour.html

Es el punto de llegada! El recorrido no ha sido tan largo.

La clase *MonoBehaviour* es la clase base de cualquier componente que se desee crear. No se hará referencia aquí a su única variable, que puede ser consultada en la documentación sobre el tema.

En cambio, sólo se abordará en esta sección una clase particular de métodos que la documentación denomina *Overridable Functions*.[1]

La clase *MonoBehaviour* presenta un conjunto de métodos que pueden sobrecargarse. Al ser muchos, sólo se listarán los más importantes pero es **muy** aconsejable leer la documentación para conocerlos a todos:

Método	Descripción
Awake	Este método será ejecutado cuando el *engine* cargue el componente. Esto significa que sólo será ejecutado una vez durante su vida. Unity aconseja no definir el constructor en los componentes. Por lo cual los métodos *Awake* y *Start* deberán ser los utilizados para la inicialización.
Start	Este método será ejecutado una sola vez antes del primer *Update*. Esto significa que sólo será ejecutado una vez durante la vida del componente. El método *Start* difiere del método *Awake* únicamente en el momento en el que es ejecutado. *Awake* será ejecutado al momento de cargarse el componente y esto ocurrirá inmediatamente después de cargada la escena en la que se encuentra. En cambio, *Start* postergará su ejecución hasta el momento antes de ejecutarse por primera vez el método *Update*. *Start* es un excelente candidato cuando se quiere postergar la inicialización de un componente hasta último momento, y de esta forma evitar demoras al momento de cargar la escena.
Update	El método *Update* se ejecutará una vez por *frame* si el *MonoBehaviour* se encuentra activo (ya se presentó antes el atributo *enabled* de la clase *Behaviour*). Se puede configurar el atributo *enabled* de la clase *Behaviour* desde el editor utilizando el *checkbox* que se encuentra al lado del nombre del componente. No se debe confundir con el *checkbox* que está al lado del nombre del *GameObject*, que configura el atributo *active* de la clase *GameObject*. Este será el lugar indicado para llevar a cabo cualquier comportamiento que se desee darle al *GameObject*.
LateUpdate	El método *LateUpdate* se ejecutará una vez por *frame* si el *MonoBehaviour* se encuentra activo. *LateUpdate* será invocado una vez que todos los métodos *Update* lo hayan sido.
OnGUI	Este método deberá ser sobrecargado para renderizar y manejar los eventos de la interfaz de usuario. *OnGUI* puede ser "llamado" más de una vez por *frame* (de hecho suele ser llamado dos pero esto no siempre es así; se verá más adelante, cuando se trabaje sobre la GUI).

[1] No se detalla la definición de este mecanismo, llamado method overriding (http://en.wikipedia.org/wiki/Method_overriding), ya que se considera que los alumnos lo conocen y manejan.

El Primer componente

Es hora de ver un ejemplo. A continuación se creará un componente que permitirá hacer girar un *GameObject* alrededor de uno de los ejes de coordenadas. Primero se deberá crear el *script* accediendo al menu *"Assets > Create > C# Script"*.

A continuación se muestra el código, que luego será analizado línea por línea.

```
01  using UnityEngine;
02  using System.Collections;
03
04  public class Rotation : MonoBehaviour {
05
06      public float degreesPerSecond = 45.0f;
07      void Start() {
08      }
09
10      void Update() {
11          transform.Rotate( 0, degreesPerSecond * Time.deltaTime, 0 );
12      }
13
14  }
15
```

El análisis comenzará por la línea *04*.

```
04  public class Rotation : MonoBehaviour {
```

Como se ha visto, todos los componentes deberán heredar de la clase *MonoBehaviour* y eso es precisamente lo que hace la línea *04*.

La siguiente línea es la declaración de una variable pública. El *Editor* de Unity mostrará cualquier variable pública que se declare en los componentes permitiendo editar su valor directamente desde el editor.

```
05  public float degreesPerSecond = 45.0f;
```

En las líneas *08* y *09* se define el método *Start* que será ejecutado por Unity antes del primer *Update*. Unity evitará invocar el método si no encuentra una definición por lo que es aconsejable no definir métodos que no se utilicen ya que, en este caso, lo invocará de igual forma aunque no se lleve a cabo ninguna acción.

El método *Update* será el corazón del script.

```
11      void                    Update()                    {
12          transform.Rotate(  0,  degreesPerSecond  *
13  Time.deltaTime, 0 );      }
```

Ahora se analizará la línea *12* en detalle. Primero se accederá a la variable *transform*, heredada de la clase *Component*. Como se ha visto en la clase *Component*, la variable *transform* es un acceso rápido al componente *Transform* del *GameObject*.

El método para hacer girar el objeto está contenido en el componente *Transform* y se llama, como es de esperar, *Rotate*. Existen algunas variantes del método *Rotate* y es recomendable consultar la documentación para ver todos los métodos que brinda la clase *Transform*.

La multiplicación de la velocidad de rotación (*degreesPerSecond*) por *Time.deltaTime* independizará la velocidad de rotación de los FPS (Frames Per Second) del juego.

Eso es todo! Es interesante ahora que los estudiantes intenten pegar el código del script en un proyecto nuevo y jugar con él; insertar un cubo en la escena (*"GameObject > Create Other > Cube"*) y arrastrarle el script e intentar usar también otros métodos del componente *Transform*, por ejemploel método *Translate*.