



Componente formativo

**Verificación y ajustes de funcionalidad**

---

**Breve descripción:**

El componente de formación presentado describe la forma de verificar el funcionamiento del prototipo digital elaborado, aplicando metodologías ágiles y herramientas propias del motor de Unity

**Área ocupacional:**

Ciencias Naturales

---

**Junio 2023**

## Tabla de contenido

Introducción.....	3
1. Diseño del ciclo de iteraciones – backlog .....	4
2. Prototipado digital con paquetes prefabricados .....	12
3. Métricas del videojuego a partir de impresiones de detalles de eventos (debug.log).....	23
3.1. Métricas de Unity .....	24
3.2. Método debug.log .....	27
4. Verificación del videojuego .....	33
4.1. Verificación basada en el modelo SCRUM .....	33
4.2. Verificación basada en el modelo SUM .....	34
4.3. Verificación y ajustes en el entorno de desarrollo (consola, modo edición) ...	36
Síntesis .....	46
Material complementario .....	47
Glosario.....	49
Referencias bibliográficas .....	50
Créditos.....	51

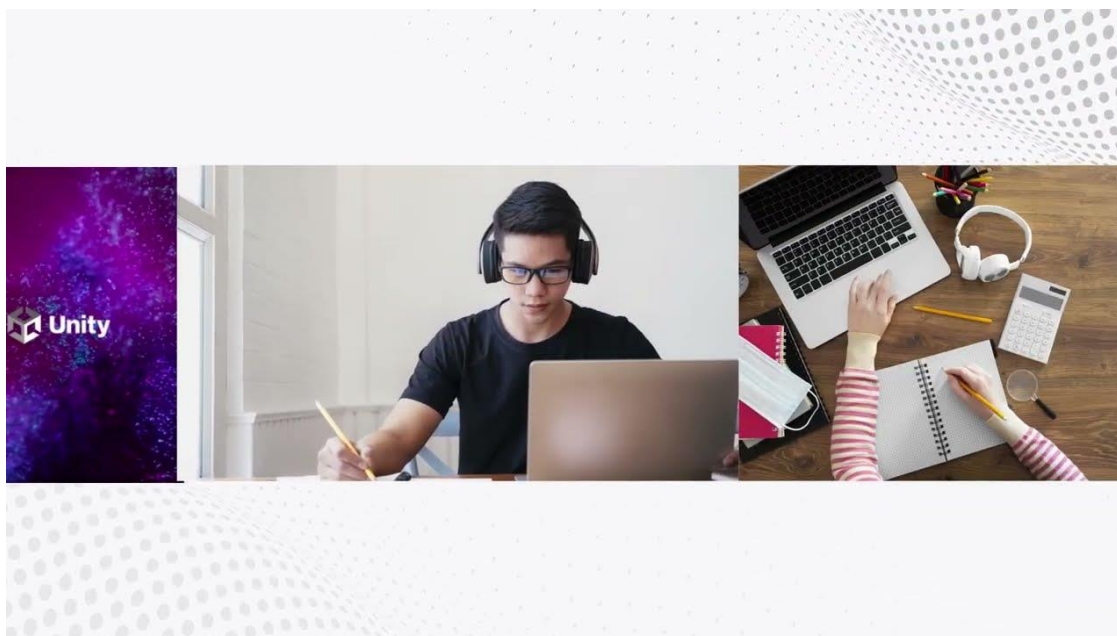
## Introducción

Uno de los aspectos más importantes para el desarrollo de los videojuegos es verificar que cumple con las funcionalidades diseñadas en el concepto gráfico. El motor de Unity contiene una función de verificación que se incorpora cuando se instala el programa y es utilizada con mayor efectividad cuando el desarrollador crea las líneas de código para comprobar una mecánica específica, igualmente posee el servicio de análisis que verifica el funcionamiento actual en aras de poder hacer las mejoras y ajustes que requiere el juego.

No obstante, es importante no solo contar con las herramientas de software sino también con las técnicas que permitan llevar a cabo comprobaciones por cada entregable del producto para ajustar inmediatamente lo que no está de acuerdo con las especificaciones iniciales o con las que van surgiendo a medida que evoluciona el producto.

En este contenido formativo se abordarán las metodologías ágiles SCRUM y SUM como técnicas para verificar con eficacia los requisitos establecidos, mecánicas o niveles de los videojuegos como se explica en el siguiente video:

### Video 1. Verificación y ajustes de funcionalidad



## Verificación y ajustes de funcionalidad

### **Síntesis del video:** Verificación y ajustes de funcionalidad

Verificar que el videojuego cumple con las especificaciones diseñadas en el concepto gráfico, es uno de los aspectos más importantes de su desarrollo, ya que allí es donde se comprueba las mecánicas definidas logran los objetivos de uso, se estudiará la verificación, bajo metodologías ágiles como SCRUM, SUM y bajo el método de consola.

Las métricas que se aplican para analizar la información clave de cómo se está ejecutando el juego a través de los servicios que presta el programa como son: Unity Analytics y Debug. Log. A partir de esto, se puede lograr un prototipo digital incremental.

¡Buen aprendizaje!

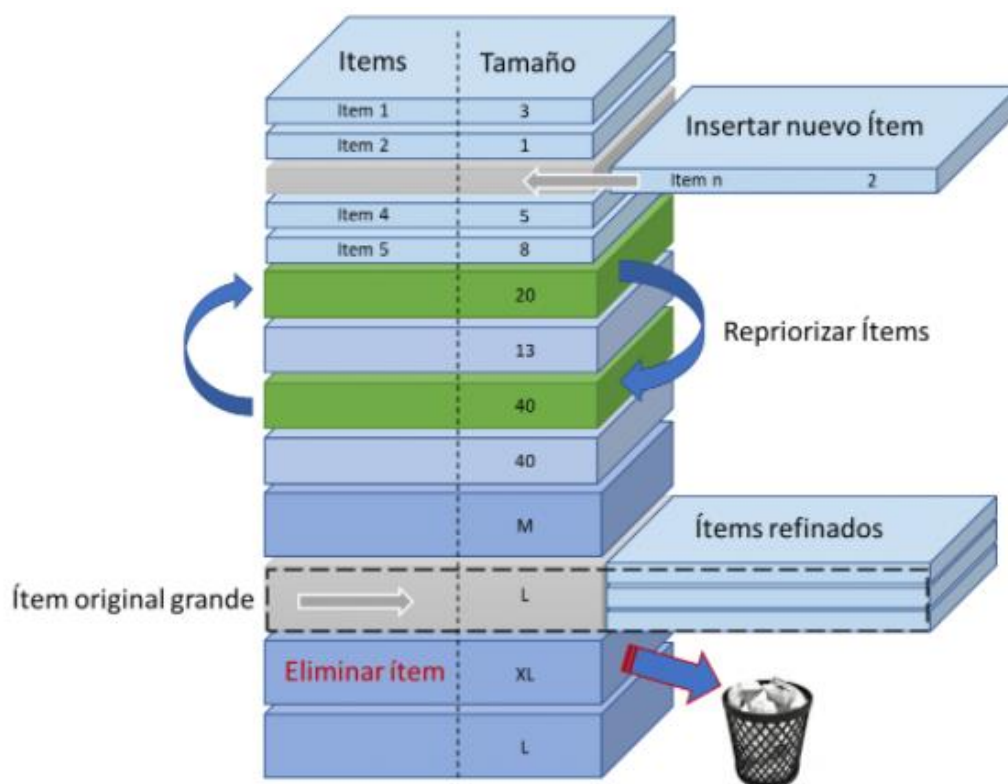
### **1. Diseño del ciclo de iteraciones – backlog**

Según García (2019) backlog es una palabra de origen inglés que significa una pila de tareas almacenadas que están esperando para ser evacuadas, igualmente, este término es usado cuando se define el ciclo de vida de un sistema de información en sus fases de análisis, diseño, desarrollo e implementación, también está asociado a metodologías de trabajo ágiles para el desarrollo y entrega de productos en menor tiempo y con mayor eficiencia.

En el contexto del método ágil el diseño de los ciclos de trabajo o iteraciones se acuña el término “Product backlog”, el cual García define como una lista ordenada y priorizada de tareas a ejecutar en un periodo de tiempo corto para completar un entregable del producto y los ciclos se repiten hasta que se obtenga el producto final.

El backlog que se observa en la Figura 1, está ordenado de acuerdo con las funciones o requisitos más importantes considerados por el cliente o dueño del producto y colocados en la parte superior del backlog, dejando en la base los de menor relevancia:

**Figura 1. Backlog**



Nota. Adaptado del Conjunto de tareas priorizadas del backlog, García (2019).

1. Durante la iteración se pueden cambiar las prioridades de las tareas según el valor que aporten al cliente las diferentes funciones que se estén desarrollando en el ítem (tarea).
2. El tamaño de los ítems (requisitos) depende del grado de detalle o refinamiento proporcionado por el cliente, ya que se deben ejecutar muy pronto, entre más pequeños sean mayor es el refinamiento o detalle, por eso los de tamaño más grande están en la parte inferior.

3. Durante el ciclo iterativo se puede decidir que un requisito no se va a ejecutar y se puede eliminar tranquilamente de la pila de requisitos o adicionar uno que no se había tenido en cuenta para completar la entrega.

El backlog se materializa en un documento que contiene los requisitos necesarios para obtener el producto final esperado por el cliente. Los aspectos que lo caracterizan se pueden resumir a continuación:

- a) Los ítems o requisitos considerados se deben colocar en una lista en orden de importancia para el cliente, siendo los primeros los de mayor relevancia.
- b) El backlog se puede modificar a lo largo del ciclo iterativo adicionando o eliminando ítems de acuerdo con las prioridades que vayan surgiendo durante el proceso de desarrollo.
- c) El refinamiento de los ítems depende de los detalles proporcionados por el cliente, siendo los más refinados aquellos que se colocan al comienzo de la lista.
- d) El tamaño de los ítems depende del grado de refinamiento (detalles) que posean, siendo los más pequeños aquellos que estén provistos del mayor número de detalles y por el contrario, los de mayor tamaño son los de menor refinamiento.
- e) Cada ítem del backlog debe asociar la función que se va a desarrollar para cumplir el requisito en un lenguaje claro y entendible por el cliente.
- f) Cada ítem del backlog debe ser independiente uno del otro para poder cambiar el orden de ejecución si así se requiere.

Cuando se desarrolla un videojuego utilizando metodologías ágiles como por ejemplo SCRUM se debe tener en cuenta:

- a) Lo primero que se diseña para empezar las tareas es el Product backlog; que en este caso sería la lista priorizada de las características principales del juego.

- b) El Product Owner (PO) es el que decide la prioridad de dichas características, ya que es el responsable ante el cliente de la calidad del producto terminado. Para establecer dicha prioridad es necesario identificar los aspectos más importantes para el jugador en su experiencia con el producto.
- c) El Scrum Team (SC) o equipo desarrollador del videojuego establecerá los ítems de la lista que se trabajarán en cada ciclo iterativo que en adelante se llamará Sprint; por lo tanto, cada Sprint es el tiempo de duración de cada ciclo de tareas que puede ser de 1 a 4 semanas, dependiendo del grado de complejidad de los ítems.
- d) Antes de comenzar cada Sprint, el Scrum Team revisa la lista de ítems a desarrollar, selecciona los ítems del Sprint definiendo su objetivo y cada persona del equipo asume una responsabilidad sobre los componentes de cada ítem que se compromete a desarrollar, colocando estos elementos en una nueva lista llamada Sprint backlog.

Cuando se desarrolla un proyecto de un videojuego, no necesariamente se hace una entrega después de terminar un Sprint; puesto que pueden necesitarse varios Sprints para lograr terminar una funcionalidad o mecánica del juego, de manera que tenga sentido para el jugador que lo va a probar.

Cada vez que se realiza una entrega se pueden determinar los bugs o errores que puede arrojar el producto cuando sea comprobado por el cliente, lo cual garantiza que antes de que esté 100% terminado debe contener el mínimo de fallos.

En el siguiente gráfico se observa el ciclo completo de un Sprint en la metodología SCRUM:

Product Owner

Se responsabiliza de elaborar con el cliente la lista de características o requisitos principales que ha de tener el videojuego o Product Backlog.

#### Development Team

El equipo desarrollador del producto, observando la lista que determina los items a ejecutar por cada Sprint, comprometiéndose cada miembro con tareas específicas por desarrollar, lo que genera una nueva lista o Sprint Backlog.

#### Scrum Master

Se asegura de que haya claridad sobre los objetivos y el alcance del producto a desarrollar e igualmente elimina cualquier impedimento para maximizar la productividad.

#### Product Backlog

Contiene la lista de los elementos que han priorizado el cliente con el Product Owner, refinando los detalles para que los desarrolladores ejecuten en el menor tiempo posible los ítems de la lista.

#### Sprint Planning Meeting

El equipo reunido realiza la planeación del Sprint determinando el tiempo que durará de acuerdo con la lista del Sprint Backlog.

#### Daily Scrum

Se realiza una reunión diariamente para asegurarse de que el proceso se está llevando a cabo satisfactoriamente.

#### Sprint review meeting

Se revisa el objetivo y el alcance del Sprint donde se verifica que las tareas que se están desarrollando, están dando cumplimiento a los ítems del Product Backlog.

#### Sprint retrospective

Cada día se realiza una retrospectiva del trabajo realizado para identificar los puntos clave en los que se puede mejorar la productividad.



## Software Increment

Al finalizar el Sprint se obtiene una parte que incrementa las características definidas para el producto final.

Para la elaboración de los Product backlog y los Sprint backlog se pueden utilizar diferentes maneras:

a. Se puede diseñar un formato para elaborar el Product backlog, el cual debe indicar claramente la prioridad y la descripción del ítem que debe desarrollar el Scrum Team, observar el ejemplo de la tabla a continuación:

**Tabla 1.** Ejemplo Product Backlog

Nombre del videojuego: La isla de los tesoros escondidos

Prioridad del ítem	Descripción
1	Interfaz principal donde se visualice la isla y los activos que la componen, colocación de los tesoros en el nivel más bajo para que no se vean.
2	Diseño del personaje principal que buscará los tesoros escondidos. Crear animaciones con acciones de correr, saltar, mirar mapa, caer, caminar.
3	Mapa de navegación que es consultado por el personaje para ubicarse en la isla.
4	Efecto del día y de la noche.
5	Diseño de animales feroces que atacan al personaje principal, movimientos y funciones de ataque.
6	Diseño del primer tesoro, caja contenedora, elementos brillantes.

Prioridad del ítem	Descripción
7	Recompensa de 300 monedas por encontrar el primero de 5 tesoros.
8	Mundo 2: otra perspectiva de la isla pasa al siguiente nivel después de encontrar el primer tesoro.

La lista puede contener muchos ítems de acuerdo con la magnitud del proyecto a desarrollar y, para una elaboración detallada se debe consultar el documento de diseño del juego, ya que es allí donde se encuentra la lógica de la historia y todos los detalles de los elementos y mecánicas que la componen.

b. También se puede diseñar el formato del Sprint Backlog con el cual el equipo de desarrollo determinará las tareas a ejecutar en cada Sprint, como se observa, en la siguiente figura:

**Figura 2. Sprint Backlog**

Objetivo Sprint #1	Realizar las funciones del mundo 1			Fecha inicio del Sprint		dd/mm/aaaa
				Fecha final del Sprint		dd/mm/aaaa
				Nº de días del Sprint		30
				Nº horas de trabajo x día		8
Ítem del Product Backlog		Prioridad en el Product Backlog	Horas estimadas totales	Horas de trabajo/día	Responsable	Estado (sin iniciar, progreso, terminado)
Interfaz principal		1				
Tarea	Estructura de contenedores		3	3	XXXXXXXXXX	T
Tarea	Selección de activos		2	2	HHHHHHHHHH	T
Tarea	Integración de activos		5	3	VVVVVVVV	P
Tarea	Configurar nivel de componentes		4	0	GGGGGGGG	S
Personaje principal		2				
Tarea	Seleccionar personaje		3	2	KKKKKKKKK	P
Tarea	Crear 6 animaciones		30	8	LLLLLLLLL	P

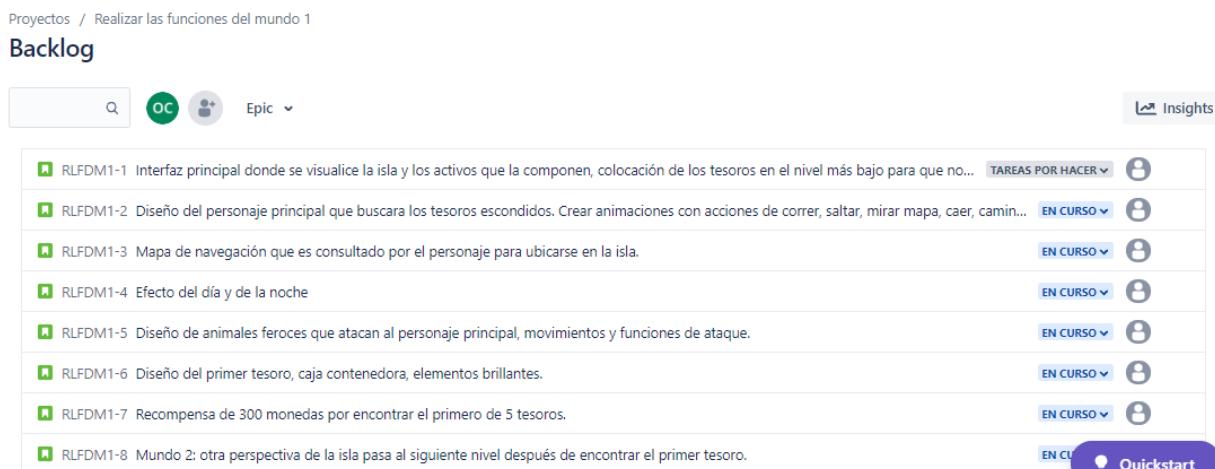
El Sprint Backlog está compuesto por tareas que son medidas en horas, teniendo en cuenta las disciplinas de las personas comprometidas en cada tarea y el número de personas con las que el equipo cuenta para llevar a cabo el Sprint.

Por ejemplo, si se deben crear 6 animaciones para el juego y solo se cuenta con una persona que se demora 8 horas por día, quiere decir que tardará más de 4 semanas para su elaboración, sobrepasando la duración del Sprint; lo que quiere decir que esta tarea se debe dividir en subtareas que harán parte de otros Sprint o en su defecto, contactar otro especialista para repartir el trabajo.

c. Se pueden utilizar herramientas digitales gratuitas como Jira Software para la elaboración de los Product Backlog y los Sprint Backlog. Las plantillas son proporcionadas por el programa:

En la siguiente figura se observa el Product Backlog del juego “La isla de los tesoros escondidos”.

**Figura 3. Plantilla Jira Software**



Nota. Backlog del juego La isla de los tesoros escondidos.



sea necesario. Los prefabs actúan como plantillas que se pueden utilizar en cualquier momento para no repetir las características de un objeto.

Se debe recordar que los asset son cualquier tipo de recurso que se utilice en el videojuego como, por ejemplo, luces, sistema de partículas, imágenes, texturas, materiales, sonidos, objetos dentro del escenario como rocas, pasto, entre otros.

## **Paquetes prefabs**

Cuando se habla de paquetes prefabricados se hace referencia a que no es necesario crear los prefabs, sino que ya el sector de la industria de videojuegos ha desarrollado una serie de objetos que pueden ser utilizados de manera libre o paga, si lo que se busca no está dentro los que se pueden descargar de manera gratuita.

Estos paquetes están agrupados por categorías de acuerdo con lo que puede requerir un juego para armar las escenas de la historia; por ejemplo, sistema de partículas, medio ambiente, vehículos, efectos especiales, terrenos, edificaciones, entre otros.

Si se está utilizando el motor de Unity para diseñar y desarrollar un juego, desde las opciones del menú el programa permite la creación de prefabs o la importación de paquetes prefabricados estándar e igualmente gratuitos o comprados de los que ofrece la tienda Unity Asset Store.

### **Crear un prefabs desde las opciones del menú de Unity**

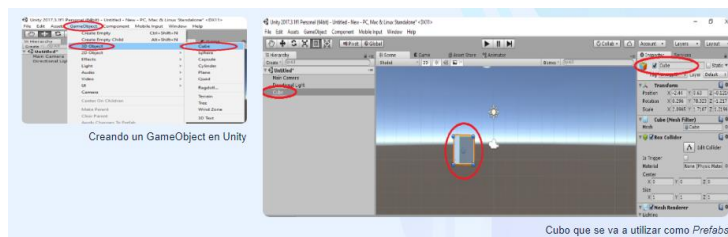
Para este prefabs se va a suponer que la escena necesita varios cubos iguales colocados en el mismo espacio, para lo cual se debe crear un prefabs del cubo de acuerdo con los siguientes pasos:

#### **1. Crear el objeto**

Desde el menú en la opción GameObject se va a seleccionar un objeto 3D que corresponde al cubo:

Menú -> GameObject -> 3D Object -> Cube

**Figura 5. Crear el objeto**

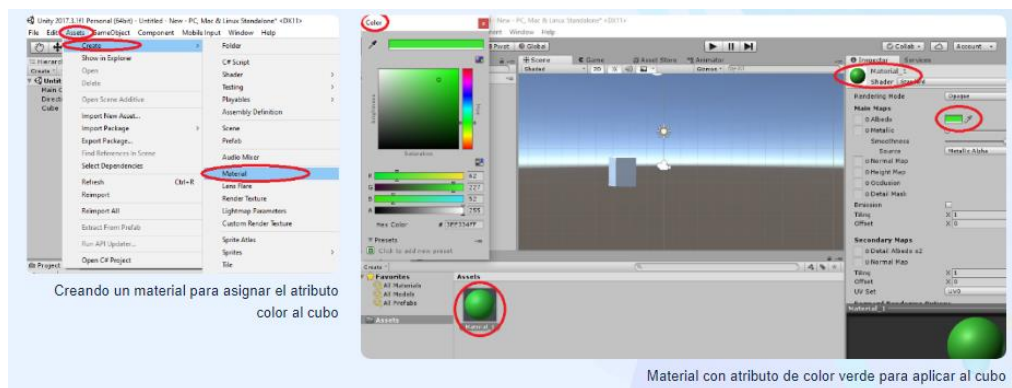


## 2. Crear un material

Ahora se le van a colocar el atributo de color verde para lo cual es necesario crear un Material:

Menú -> Asset -> Create -> Material

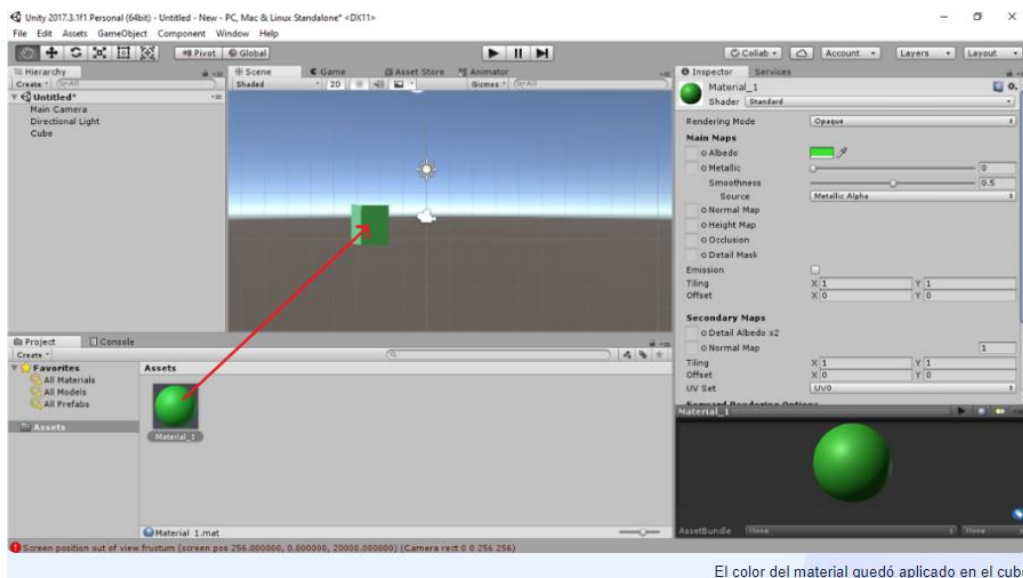
**Figura 6. Crear un material**



## 3. Aplicación de color

Arrastrar con el mouse el material creado hasta el cubo para que se aplique el color.

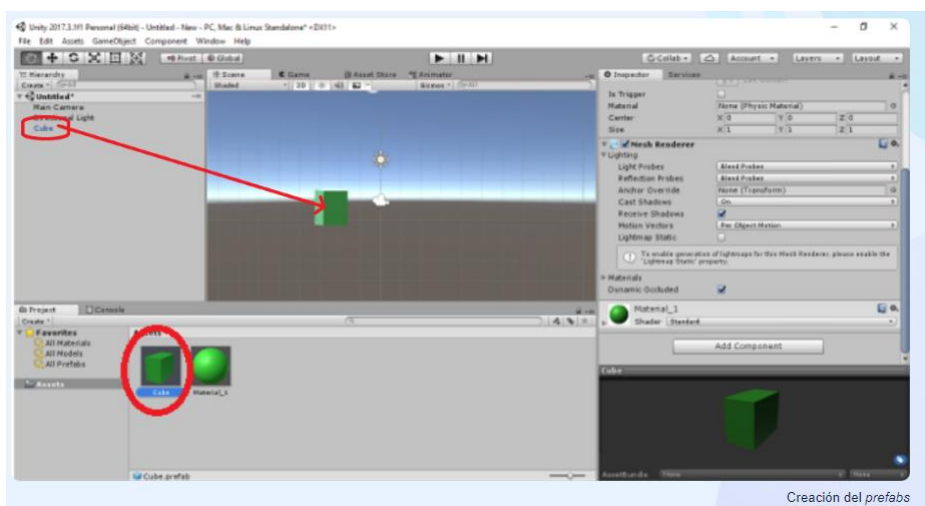
**Figura 7. Aplicación de color**



## Crear el *prefabs*

Hacer clic sostenido en la palabra cube que aparece en la ventana de Jerarquía y soltarla en la ventana de proyecto, lo cual da como resultado el prefabs que se distingue del material porque tiene la forma del cubo y el nombre está resaltado en azul.

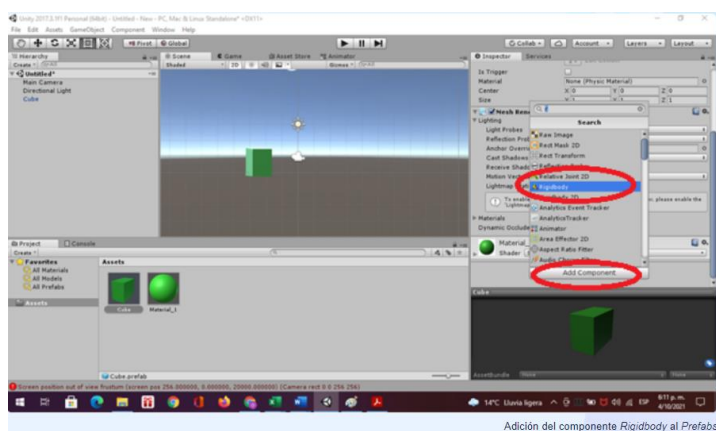
**Figura 8. Crear el *prefabs***



## Dar atributo *Rigidbody*

Al prefabs se le puede incorporar este atributo que significa que se puede mover libremente si se desea cuando se desarrollen líneas de código.

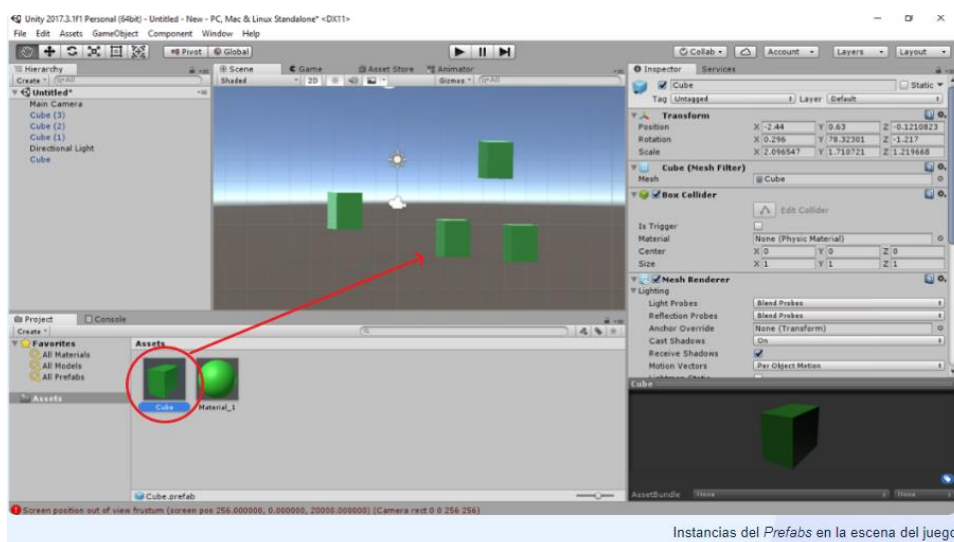
**Figura 9.** Dar atributo *Rigidbody*



## Hacer repeticiones del *prefabs*

Una vez creado el prefabs se pueden hacer cuantas repeticiones o instancias se necesiten de este elemento dentro de la escena; solamente arrastrando el prefabs a la ventana de la escena del juego.

**Figura 10.** Hace repeticiones del prefabs





## Importar un paquete prefabricado estándar

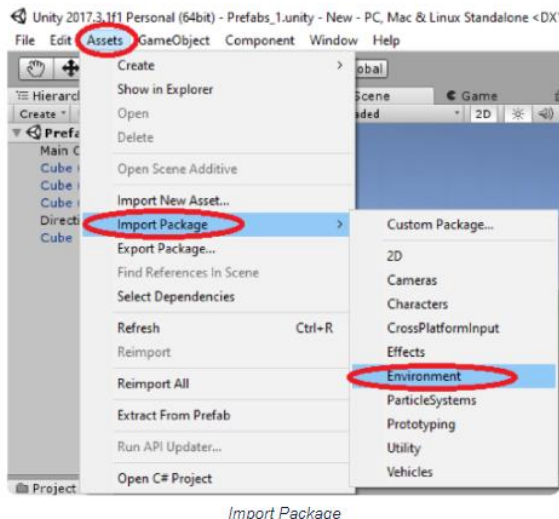
El motor de Unity tiene incorporados asset que actúan como recursos prefabricados llamados activos estándar, los cuales facilitan la creación de una escena, proporcionando elementos tales como medio ambiente, personajes y sistemas de partículas entre otros:

### Importar paquete

Cuando se ha creado el proyecto en el programa Unity se podrá incorporar cualquiera de estos asset accionando del menú la opción Assets, clic en importar paquete y seleccionar uno de la lista desplegable; en este caso se incorporará el paquete de medio ambiente:

Menu -> Assets -> Import Package -> Environment

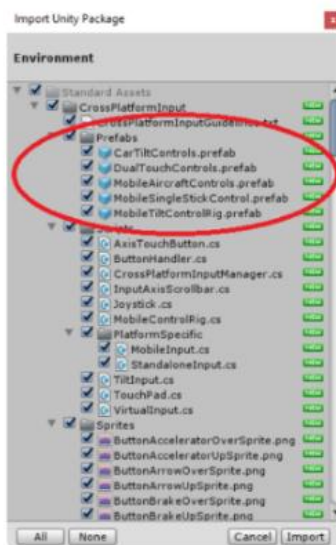
**Figura 11.** Importar paquete



### Prefabs Environment

Cuando se ha seleccionado el paquete, el programa muestra todos los archivos que importará al proyecto una vez se haga clic en el botón Import.

**Figura 12. Prefabs Environment**

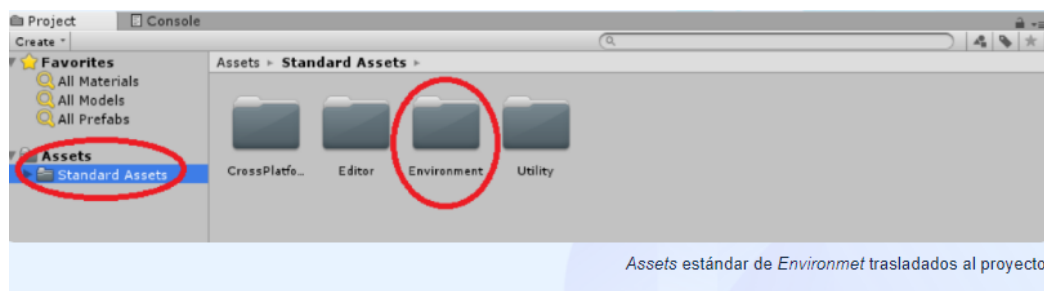


*Prefabs de medio ambiente que se incorporarán en el proyecto.*

## Assets en carpeta

En la figura se pueden observar los activos prefabricados que se trasladaron a la carpeta de asset del proyecto del videojuego en la ventana Project.

**Figura 13. Assets en carpeta**

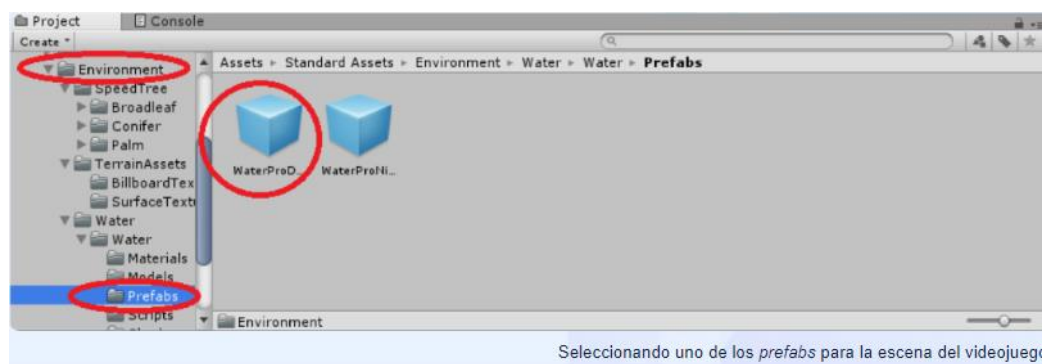


*Assets estándar de Environment trasladados al proyecto.*

## Prefabs WaterProD

Al hacer clic en la carpeta Environment se selecciona la carpeta de prefabs y se observan dos recursos que tienen que ver con agua. Se puede seleccionar cualquiera, por ejemplo, el primero.

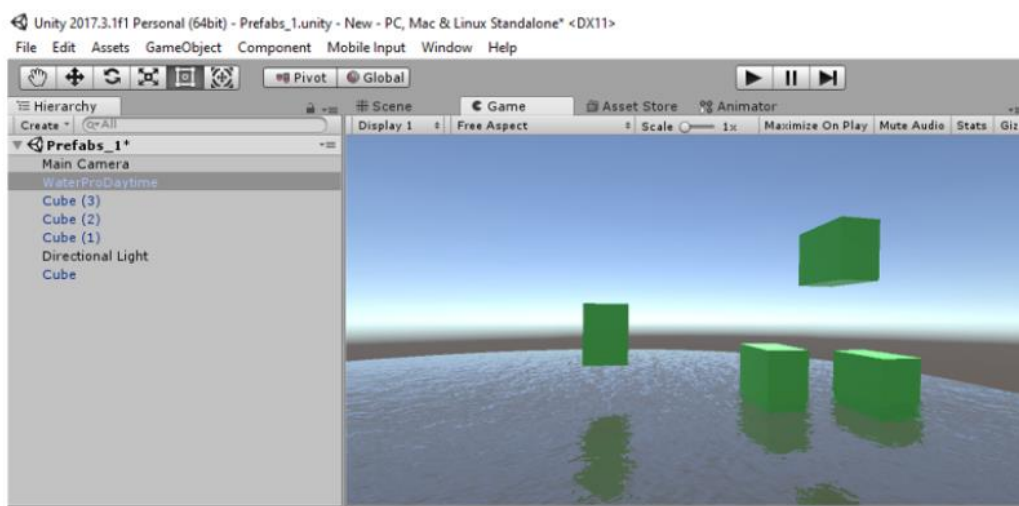
**Figura 14.** Prefabs WaterProD



## Escena de agua

Si se arrastra el prefabs seleccionado al espacio de la escena donde se copiaron los cubos anteriormente; entonces estos quedan sobre el agua como se observa en la figura 15.

**Figura 15.** Escena de agua



GameObjects sobre agua

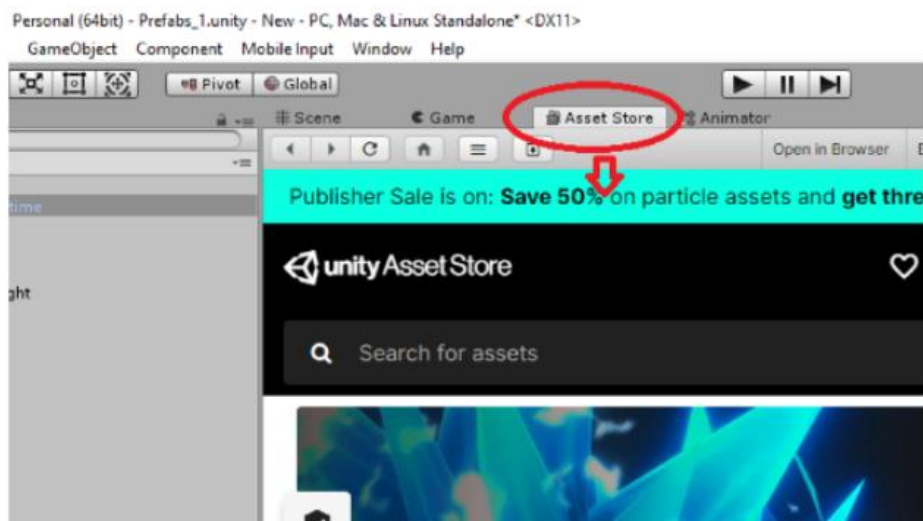
## Importar paquete de Asset Store

Cuando en los paquetes estándar no están los recursos necesarios para el videojuego se pueden importar de la tienda de Unity, esta acción se puede realizar estando en el motor de Unity:

### Tienda Unity

Haciendo clic en la pestaña Asset Store e inmediatamente el programa se conecta en línea con la tienda.

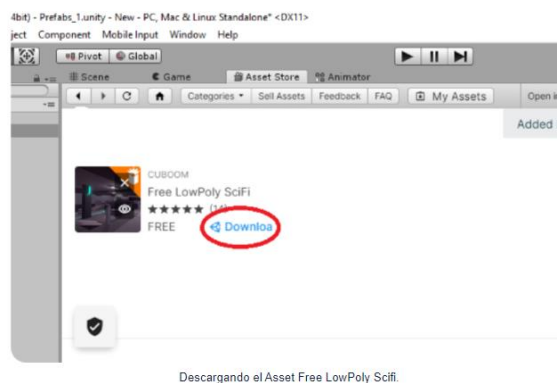
**Figura 16. Tienda Unity**



### Asset Free

En este caso se seleccionó un paquete de ciencia ficción de descarga gratuita llamado LowPoly SciFi, el cual se debe descargar haciendo clic en el botón Download.

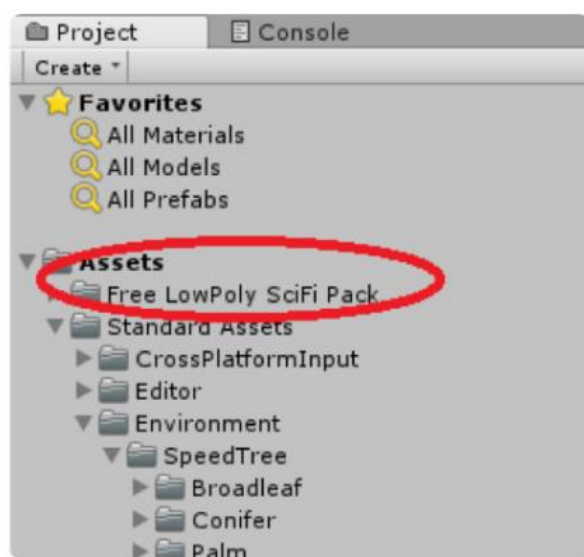
**Figura 17. Asset Free**



## Adición de Assets Free

Cuando el paquete ha sido descargado e importado se adiciona a la carpeta de Assets del proyecto de videojuego que se está desarrollando.

**Figura 18.** Adición de Assets Free



LowPoly SciFi adicionado a la carpeta de Assets en la ventana Project.

## Prototipo

A continuación, se observa un prototipo diseñado con los prefabs del paquete importado LowPoly SciFi Pack, los cuales se pueden disponer en escena de acuerdo con las necesidades del videojuego y la lógica de las mecánicas.

**Figura 19. Prototipo**



Armado de un prototipo con los Prefabs de LowPoly SciFi Pack.

Para profundizar en el tema consultar en el material complementario los siguientes artículos: Instanciar un objeto en Unity 3D y Prefabs.

### **3. Métricas del videojuego a partir de impresiones de detalles de eventos (debug.log)**

Es un conjunto de herramientas de análisis que proporcionan información acerca de la experiencia que tienen los usuarios con el uso del producto y de esta forma medir su calidad. Esta información puede ser utilizada para realizar mejoras en todos los aspectos del juego.

A grandes rasgos la información que se puede extraer de un videojuego es:

#### **Datos de los usuarios del juego**

Se refiere a los datos de las personas que ingresan al juego, como por ejemplo, nombre, edad, género, lugar de nacimiento, entre otros. Esta información permite segmentar

a los jugadores de acuerdo con su perfil para tomar decisiones respecto al desarrollo de funciones que despierten su interés.

### **Datos de hardware y software utilizados**

Se refiere a los datos que se puedan obtener de los dispositivos, sistemas operativos e infraestructura que posean los usuarios para ejecutar el juego, puesto que de esta parte técnica depende también su experiencia de uso. No se puede lanzar un producto de videojuego en un lugar donde no funcionan adecuadamente las comunicaciones.

### **Datos de jugabilidad**

Hace referencia a la experiencia del usuario en las diferentes sesiones del juego, se analiza la comprensión de las mecánicas a través de las diferentes sesiones jugadas, el recorrido por los niveles, cuántos logra superar, el tiempo que demora en avanzar y el tiempo total que invierte en el juego por día, semana y mes.

## **3.1. Métricas de Unity**

El motor de videojuegos Unity utiliza el sistema Analytics, servicio que se configura automáticamente al ser instalado el programa y que contiene las siguientes métricas:

### **Métricas del jugador**

Proporciona información sobre el número de jugadores que inician una sesión durante las 24 horas del día; con lo cual se pueden obtener las estadísticas del porcentaje mensual de participantes activos tanto nuevos como los que regresan.

### **Métricas de sesión**

Se refiere al número total de sesiones que juegan en promedio los usuarios por día y el tiempo de juego promedio de estos.

### **Métrica de retención**

Proporciona el porcentaje de jugadores que regresan al juego al siguiente día de jugar por primera vez, los que regresaron a los siete días y los que regresaron a los 30 días.



## **Métrica de monetización**

Hace referencia a los ingresos promedio por usuario que juegan en un día, los cuales pueden ser obtenidos de los anuncios. Para esto Unity hace la verificación mediante la función Analytics Transaction.

## **Métrica de Ads**

O métrica de anuncios, hace referencia a los ingresos promedios por anuncios que ven los jugadores durante el juego, al igual que los ingresos totales. Los anuncios de video que se reproducen en el juego y los ingresos estimados por 1000 impresiones de anuncios.

## **Generador de segmentos**

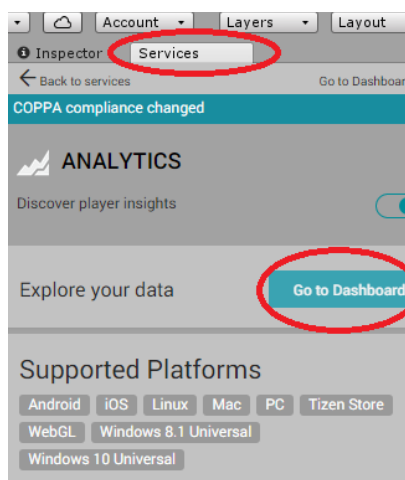
Dentro de la base de datos de los jugadores, Unity forma grupos diferenciados aspectos como:

- a) Segmentos del ciclo de vida: número de días que jugaron desde que ingresaron por primera vez: (1-3) días; (4-7) días; (8-14) días; (7-30) días; (31 -90) días.
- b) Segmentos geográficos: de acuerdo con la dirección IP desde donde se conecta el jugador Unity lo geolocaliza. El desarrollador puede definir las zonas geográficas o países que va a incluir.
- c) Segmentos de monetización: agrupa los jugadores de acuerdo con la cantidad de dinero que gastan en el juego.
- d) Segmentos por plataforma: clasifica a los jugadores de acuerdo con dos plataformas, los que utilizan Android o los que utilizan IOS.
- e) Segmentos personalizados: es una categoría de jugadores que se crea combinando reglas de los segmentos anteriores.

Para hacer uso de las métricas de Unity el juego debe estar por lo menos en la versión Beta, en la cual se hacen pruebas de usuario para medir su experiencia. Los pasos a seguir son:

1. Los desarrolladores deben abrir el proyecto del videojuego y en la ventana del inspector:

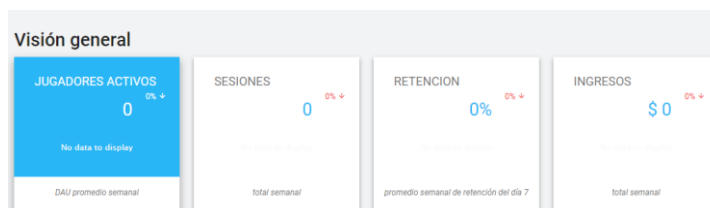
**Figura 20.** Servicio de análisis del videojuego



2. Una vez cargada la pestaña de servicios de Unity se hace clic en el botón ir al tablero: Go to dashboard para acceder a las diferentes métricas de análisis.

Las métricas que se observan en la figura, aparecen en (0), debido a que el juego se está desarrollando y todavía no se han convocado usuarios para comprobar las funciones 1.

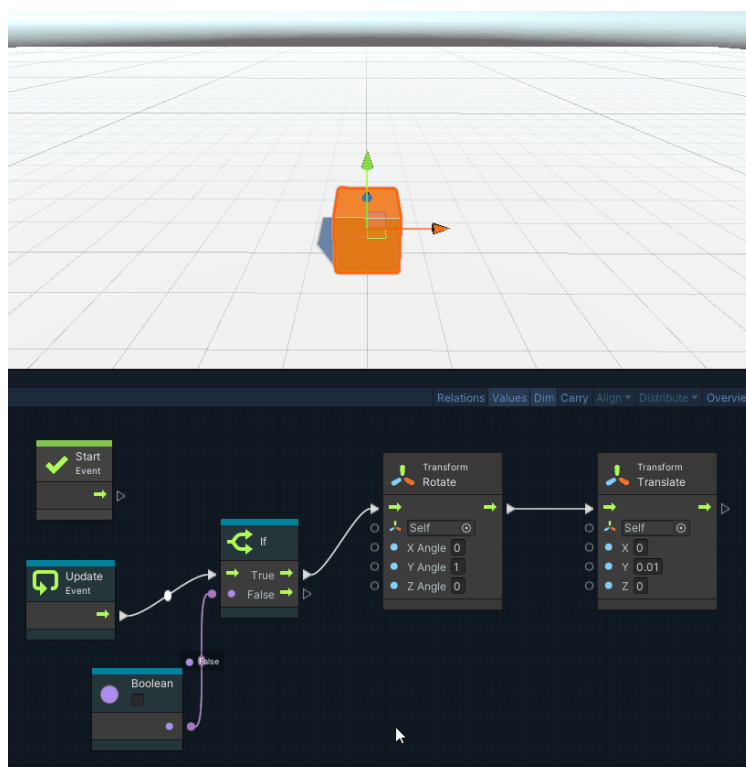
**Figura 21.** Métricas de un juego en fase de diseño



### 3.2. Método debug.log

En Unity el método debug.log consiste en una función que se utiliza para imprimir mensajes en la ventana de la consola con el objeto de visualizar los eventos del juego, en cuanto al comportamiento de los objetos y componentes que se han colocado en las escenas, inspeccionando el flujo de las acciones con el propósito de depurarlas, por ejemplo, se va a inspeccionar el evento A: rotación del cubo en la escena del juego de acuerdo con las coordenadas iniciales (X=0, Y=1, Z=0) a las coordenadas finales (X = 0, Y = 0.01, Z = 0), como se muestra a continuación:

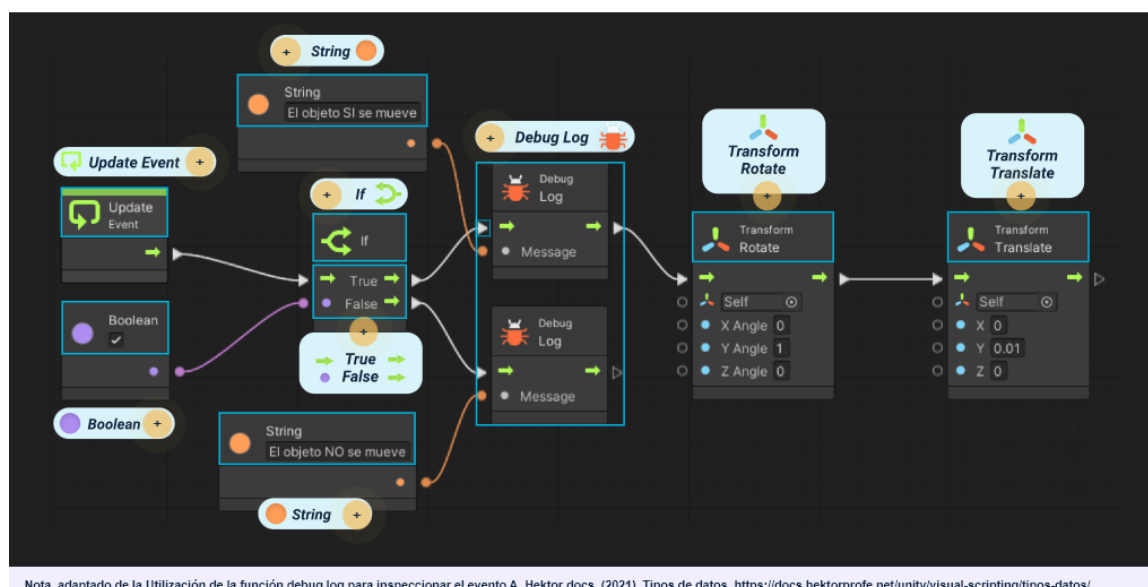
**Figura 22.** Evento A



Nota. Flujo del evento A. Hektor docs, (2021). Tipos de datos. <https://docs.hektorprofe.net/unity/visual-scripting/tipos-datos/>

## Función debug.log

El flujo de transformación se puede imprimir en la ventana de la consola de Unity a través de la función debug.log, las líneas de código relacionadas con el evento.



**String.** Corresponde a datos de tipo carácter, es decir, la escritura del mensaje “El objeto Sí se mueve” o “El Objeto NO se mueve”.

**Update.** Corresponde al modo de actualización de los eventos o acciones del juego.

**Boolean.** Es un tipo de dato lógico que puede asumir dos valores: True o False.

Se cumple la condición si el valor de la variable Boolean es **True**, si es **False** no ocurre nada.

**If.** Cumplimiento de una condición.

Para este caso del evento A, la condición es que la variable sea verdadera para que el cubo haga la rotación; de lo contrario no rota.

La función **debug.log** imprime el mensaje correspondiente en la consola, dependiendo del comportamiento del evento.

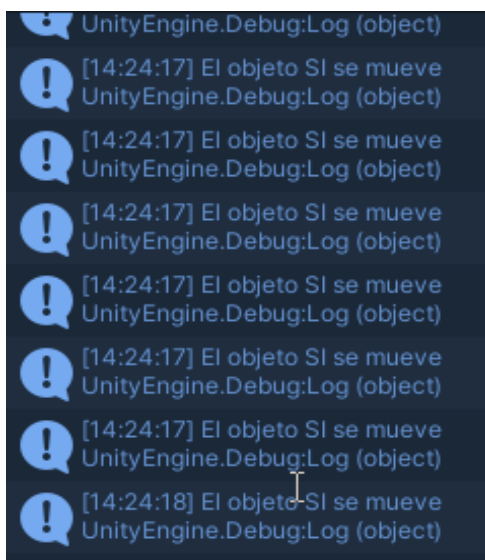
**String.** Corresponde a datos de tipo carácter es decir la escritura del mensaje ““El objeto Sí se mueve” o “El Objeto NO se mueve”.

**Transform Rotate.** Hace referencia a los datos de las coordenadas iniciales o de partida del objeto.

**Transform Translate.** Hace referencia a los datos de las coordenadas finales o de llegada del objeto.

Obsérvese en la figura que el mensaje en la ventana de la consola indica que el cubo efectivamente ha rotado de una coordenada inicial a una final: “el objeto SI se mueve”:

**Figura 23.** Mensajes consola



Nota. Impresión de mensajes del evento A en la ventana de la consola. Hektor docs, (2021). Tipos de datos. <https://docs.hektorprofe.net/unity/visual-scripting/tipos-datos/>

## Crear Script

Cuando se necesita manipular el objeto e imprimir el flujo del evento con mensajes propios se debe crear el Script mediante los siguientes pasos:

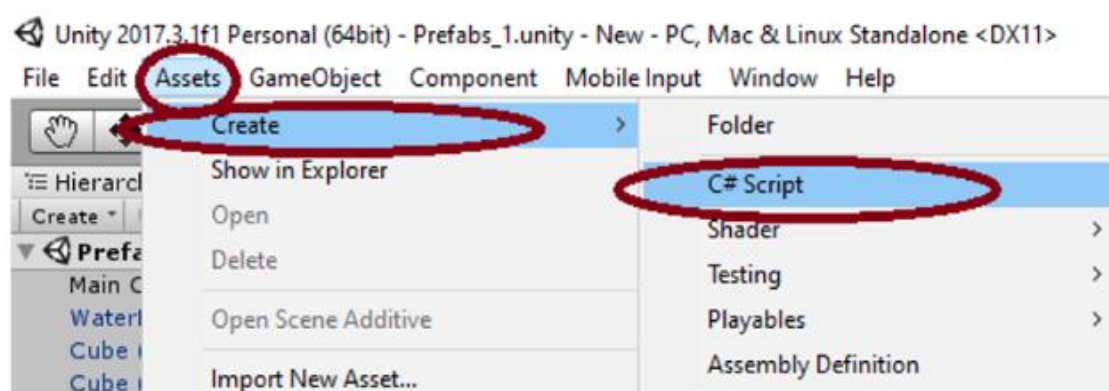
### Crear C# Script

Ingresa al menú Assets, dar clic en Create y elegir la opción C# Script:

Menu -> Assets -> Create -> C# Script

Al crear el archivo C# Script y colocarle el nombre se abre la ventana del Inspector con las líneas de código, que por defecto coloca el programa Unity y que corresponden a la inicialización del programa o Start, la actualización de los objetos y componentes Update.

**Figura 24.** Crear C# Script



Creando un *Script* para imprimir mensajes del evento

### Script mensaje

Al Script se le ha colocado el nombre Mensajes, el cual aparece en la parte superior de la ventana del Inspector con el icono de archivo C# y en la parte inferior de la ventana aparecen las líneas de código que se han escrito para imprimir los mensajes en la consola.

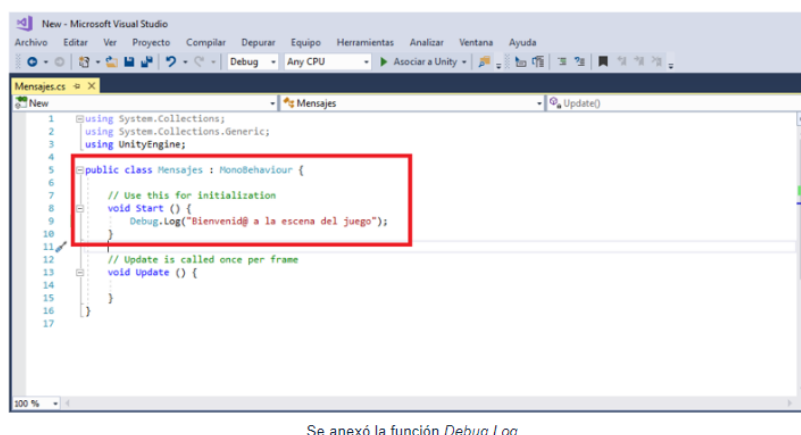
**Figura 25. Script mensaje**



## Código del Script

Para visualizar mejor las líneas de código se abre el editor Visual Studio y se escriben las líneas de código. Se observa public class Mensajes, que indica que se ha creado un Script que puede ser utilizado sin restricción en diferentes GameObjects del juego, porque se ha declarado de clase pública. La línea de código Debug.Log("Bienvenid@ a la escena del Juego") indica que se imprimirá este texto en la consola al aplicar el Script a un GameObject de la escena y se active el juego.

**Figura 26. Código del Script**



## Aplicar Script

Para aplicar el Script a la escena se debe seleccionar uno de los objetos; en este caso se seleccionó el primer cubo y después se arrastra con el mouse sostenido el Script Mensajes hacia la ventana de jerarquía al texto Cube. Después se activa el script haciendo clic en el botón avanzar.

**Figura 27. Aplicar Script**

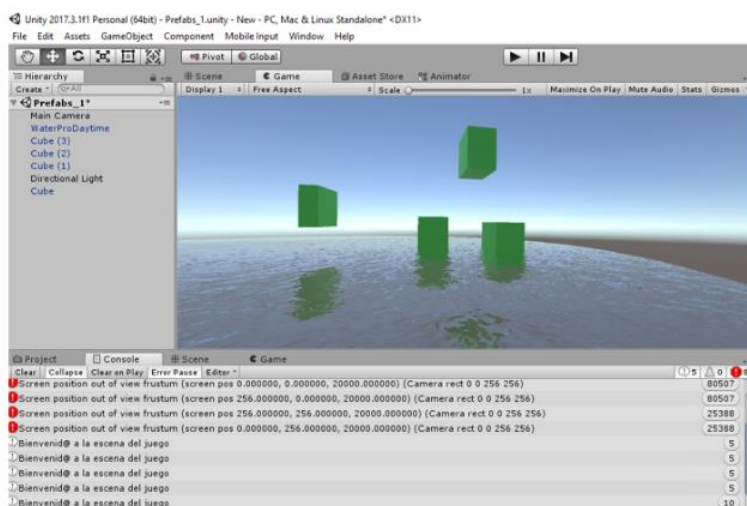


## Impresión de evento

En la figura se puede observar la impresión del mensaje del evento en la consola precedido de un icono blanco, los iconos rojos indican un error en la colocación de la cámara en la escena que puede corregirse ubicándola de nuevo.



**Figura 28.** Impresión de evento



*Debug.log que imprime la bienvenida al juego*

Para profundizar en el tema consultar en Unity Documentation: “Métricas, segmentos y terminología de Analytics” y “Depurar” que se encuentran en el material complementario.

## 4. Verificación del videojuego

Para que la comprobación y los ajustes de un videojuego se realicen rápidamente es importante hacer uso de un método ágil, que permita un trabajo fluido y un incremento del proyecto a través de cada entregable aprobado por el cliente.

### 4.1. Verificación basada en el modelo SCRUM

Al utilizar un método como SCRUM se deben planear no solo el diseño y desarrollo sino también la verificación del producto, llevando a cabo las diferentes fases del método:

Modelo Scrum: [Infografía](#).

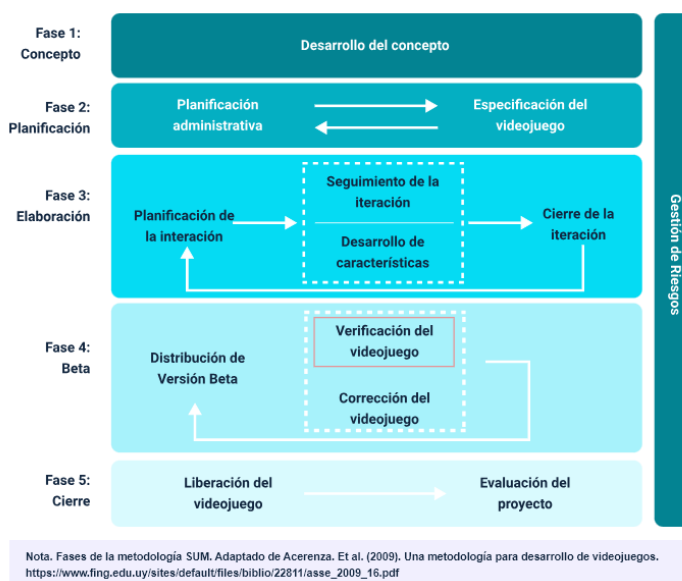
Para ampliar los conocimientos relacionados con este tema consultar el artículo “Metodología Scrum” que se encuentra en el material complementario.

## 4.2. Verificación basada en el modelo SUM

Bajo la metodología SUM el videojuego se desarrolla en cinco fases donde en cada una se debe lograr un objetivo específico, realizando un proceso de mejora continua para obtener resultados administrando eficientemente los recursos y los riesgos.

La etapa Beta es la primera versión completa del videojuego y en la siguiente figura se observa que en la “Fase 4: Beta” se encuentra la “Verificación del videojuego”, puesto que ya las funcionalidades se pueden probar:

**Figura 29. Metodología SUM**



La fase 4 está constituida por los siguientes elementos:

### Verificadores Beta

Son los designados para hacer la verificación de las funciones del videojuego, pueden tener experiencia en probar el software de videojuegos, o no poseerla, haber jugado o no.

## **Cliente**

Está en casi todas las etapas del diseño y desarrollo del videojuego, puesto que se encarga de definir el concepto del juego, especificar y priorizar las características y las tareas que le dan más valor, evalúa el producto obtenido al finalizar cada iteración, prioriza la corrección de los errores y valida las versiones del producto.

## **Equipo de desarrollo**

Está conformado por diseñadores estructurales, diseñadores gráficos y programadores. Aportan a la definición del concepto, realizan la estimación del tiempo, definen las tareas, las asignan y realizan para producir el videojuego. Evalúan el proceso y colocan los correctivos a los problemas.

## **Aspectos por verificar**

Son las características funcionales y no funcionales que deben comprobar los verificadores Beta.

## **Videojuego**

Es la versión ejecutable del contenido que es reproducida para la verificación de los aspectos funcionales y no funcionales.

## **Evaluación y errores encontrados**

Es la comprobación del funcionamiento del juego, en la cual se genera una lista de los aspectos a corregir, planificándose una iteración para priorizar las correcciones a los errores encontrados.

## **Lista de cambios priorizados**

Es la lista de errores encontrados en el funcionamiento del juego y generada en la evaluación. Los errores se deben ordenar de acuerdo con el nivel de importancia y afectación en las funcionalidades del juego.

## **Realizar ajustes**

Es la corrección de los errores que se encuentran en la lista de cambios priorizados, teniendo en cuenta la prioridad y los costos del ajuste a realizar.

Para los conocimientos en el tema consultar el documento SUM para el desarrollo de videojuegos que se encuentra en el material complementario.

#### 4.3. Verificación y ajustes en el entorno de desarrollo (consola, modo edición)

La verificación de un videojuego desarrollado en el motor Unity consiste en la comprobación de la calidad y el funcionamiento de sus mecánicas, así como de los elementos y los recursos utilizados en él. Para ello, el programa tiene incorporado un paquete Package Manager que puede utilizar el desarrollador para realizar las pruebas que necesita.

##### Video 2. Verificación y ajustes



[Verificación y ajustes](#)

##### Síntesis del video: Verificación y ajustes

¿Cómo funciona la verificación y ajustes de un videojuego desarrollado en Unity? Vea en qué consiste.

En cada fase del desarrollo de un videojuego se debe implementar la verificación para evaluar cada entregable y ajustar lo que sea necesario; de tal forma que en la última fase el proyecto esté a punto.

Las pruebas que permiten verificar el juego se llaman pruebas unitarias porque permiten analizar fragmentos de código u objetos de la escena por unidad y todos por separado.

El sistema incorporado de prueba de Unity va verificando y arrojando en la consola los errores que se pueden encontrar en cada uno de los elementos analizados. Este sistema incorporado se llama Test Runner y se puede ejecutar de dos maneras:

Primero, cuando el juego está en modo de edición: las verificaciones se hacen para probar que el juego realiza correctamente cálculos matemáticos; por ejemplo, en el número de vidas del jugador, puntajes que se deben actualizar o número de partículas que expelle el GameObject.

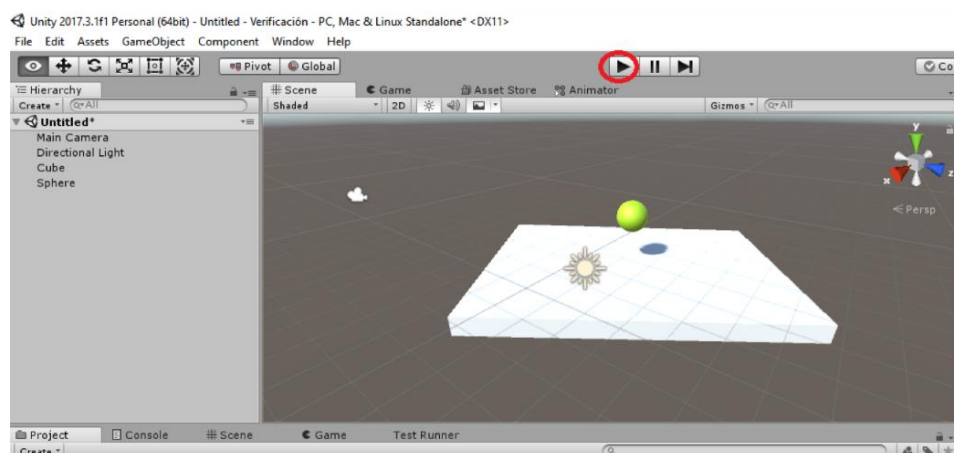
Y segundo, cuando el juego está en modo de reproducción o Play. La mayoría de los test de verificación en Unity se hacen en este modo runtime o tiempo real, puesto que se necesita que el personaje o el objeto esté en movimiento o llegue a cierta posición y de esta forma probar la mecánica del juego.

Los desarrolladores pueden crear pruebas para testear cualquier cosa en los objetos que han colocado en una escena del juego. En el siguiente ejemplo se observa una escena que contiene una esfera de color verde encima de un cubo de color blanco que hace de piso; se quiere verificar que la esfera caiga bajo el efecto de la gravedad sobre el piso que es el cubo, mediante los siguientes pasos:

### **Esfera sobre cubo**

Se quiere verificar con la función de testeo que la esfera posee el efecto de gravedad para que caiga sobre el cubo cuando se reproduzca la escena con el botón Play encerrado en el círculo rojo.

**Figura 30.** La esfera está sobre el cubo suspendida en el aire

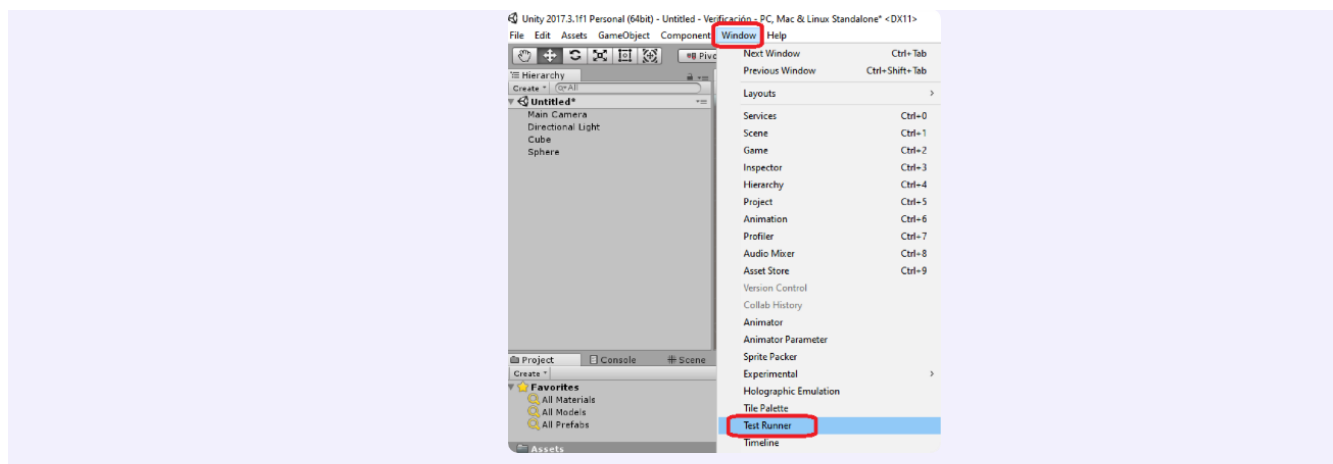


## Testear

Se va a escribir una prueba unitaria (Test) ayudado de la función Test Runner de Unity para hacer la verificación. Para ello debe activar la función de la siguiente manera:

- Menú -> Window -> Test Runner

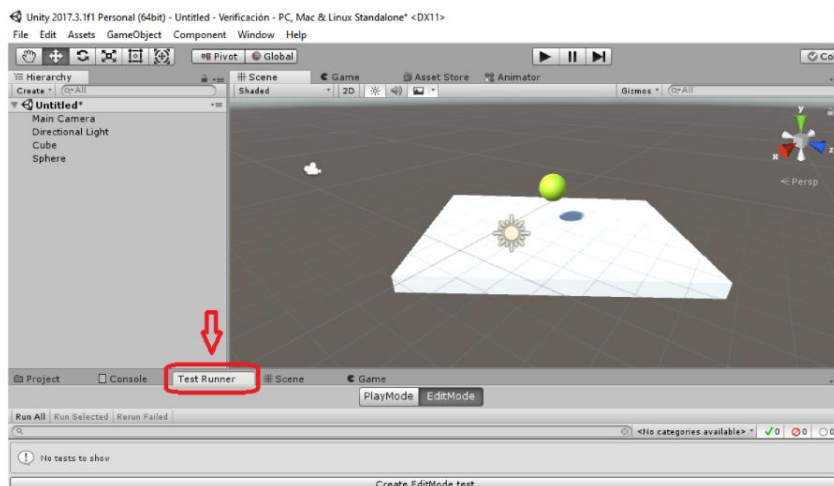
**Figura 31.** Testear



## Ventana Test Runner

Una vez abierta la ventana de Test Runner se puede arrastrar y ubicar en la parte inferior dentro del espacio de la ventana de proyecto.

**Figura 32.** Ventana Test Runner

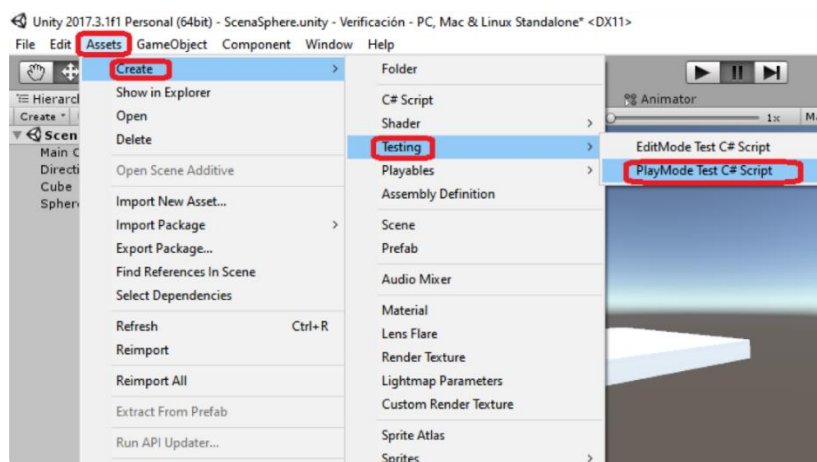


## Script de Test

En la ventana de Project se crea una carpeta que va a contener el Script donde se van a escribir las líneas de código para verificar que la esfera caiga bajo el efecto de la gravedad sobre el piso que es el cubo. Se va a realizar la prueba en modo play, por lo tanto, se coloca Test Runner en PlayMode y se crea el archivo de la siguiente manera:

- Menú -> Assets -> Create -> Testing -> PlayMode Test C# Script.

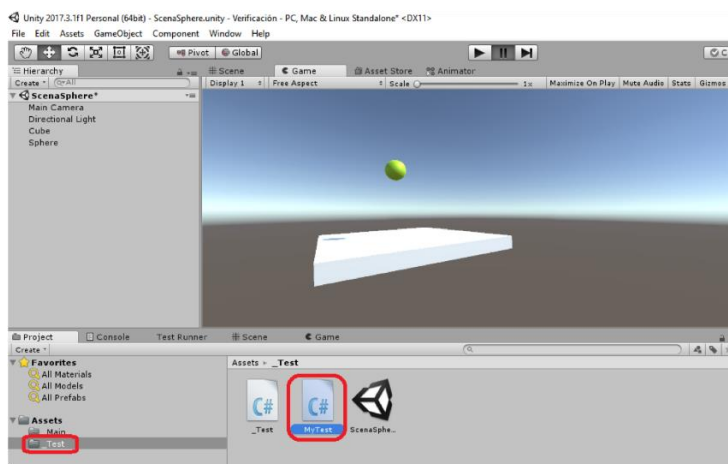
**Figura 33. Script de Test**



## Script MyTest

A la carpeta que se creó se le dio el nombre `_Test` y al Script el nombre `MyTest` tal como se muestra en la figura.

**Figura 34. Script MyTest**



## Visual Studio

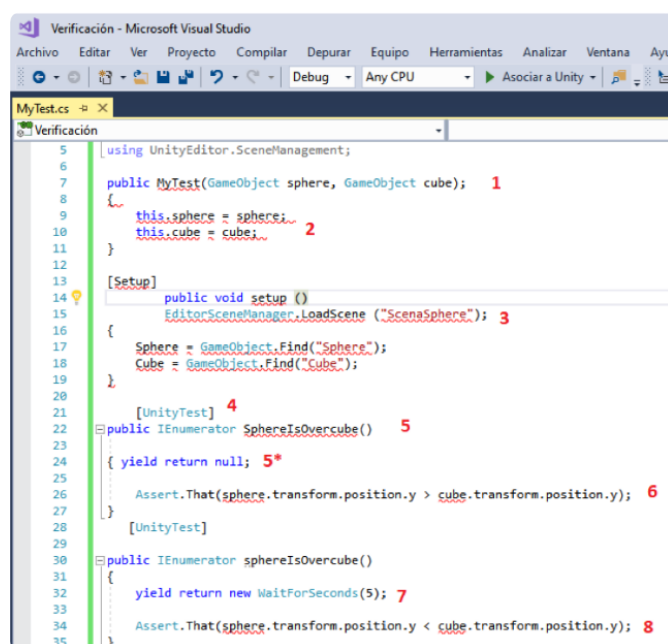
Las líneas de código del archivo se escriben en el editor de texto de Unity Visual Studio. El Script lo que pretende es que la esfera caiga a los 5 segundos de reproducir la escena en modo Game haciendo clic en el botón play del programa.



En las líneas de código marcadas en los puntos:

- El espacio MyTest se declara público con los GameObject sphere y cube.
- Se declaran las variables sphere y cube que más adelante tomaran el valor de las posiciones.
- Se carga la escena llamada "ScenaSphere".
- Indica que el Script se va a ejecutar en PlayMode en el Test Runner.
- La sphere está sobre el cube - 5 \* mientras la esfera no cambie la posición en el eje y el valor de la variable es nulo.
- Si el valor actual de la posición y de la sphere es mayor que el valor y de la posición del cubo, la esfera no cae.
- La esfera cambia la posición en el eje y el valor de la variable es 5.
- Si el valor actual de la posición y de la sphere es menor que el valor y de la posición del cubo la esfera cae.

**Figura 35.** Visual Studio



```

5 | using UnityEditor.SceneManagement;
6 |
7 | public MyTest(GameObject sphere, GameObject cube); 1
8 | {
9 |     this.sphere = sphere;
10 |    this.cube = cube; 2
11 | }
12 |
13 | [Setup]
14 | public void setup ()
15 | {
16 |     SceneManager.LoadScene ("ScenaSphere"); 3
17 | }
18 |
19 | [UnityTest] 4
20 | public IEnumerator SphereIsOvercube() 5
21 | {
22 |     yield return null; 5*
23 |
24 |     Assert.That(sphere.transform.position.y > cube.transform.position.y); 6
25 | }
26 | [UnityTest]
27 | public IEnumerator sphereIsOvercube()
28 | {
29 |     yield return new WaitForSeconds(5); 7
30 |
31 |     Assert.That(sphere.transform.position.y < cube.transform.position.y); 8
32 | }
33 |
34 |
35 |

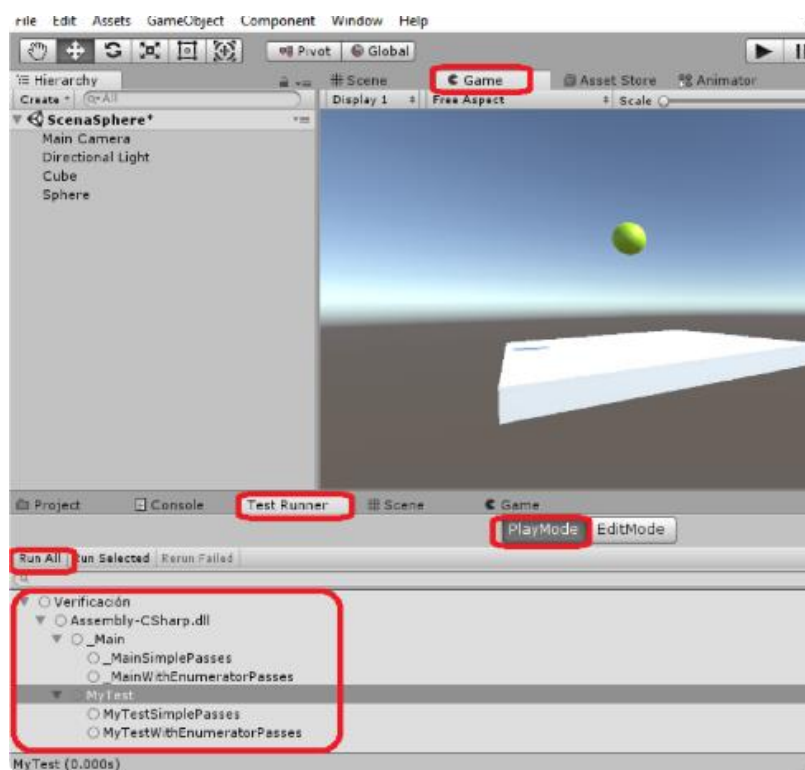
```

## Activar Test

Para correr el Test dirijase al motor Unity a la ventana de Test Runner y observe que el programa a adicionado archivos que necesita para ejecutar el Script; la escena debe estar

en modo Game, Test Runner debe estar en PlayMode y simplemente se debe hacer clic en Run All.

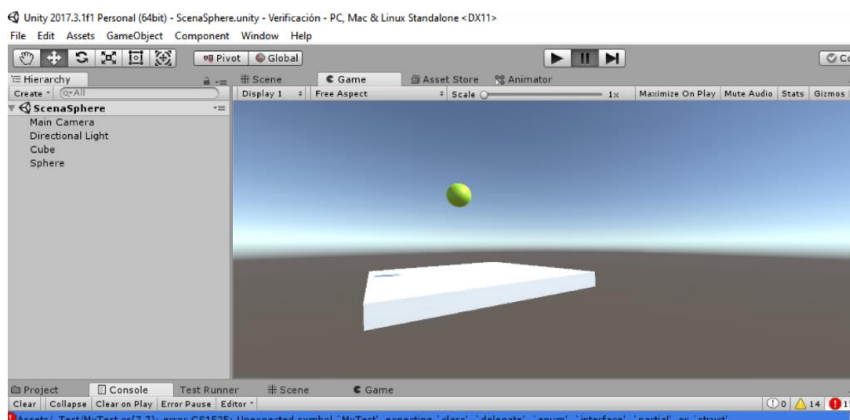
**Figura 36. Activar Test**



## Mensaje de error

Al ejecutar Run All; en la Consola aparece un error en la línea 7 del Script para ello, se debe ir al editor y resolverlo para volver a ejecutar el Test.

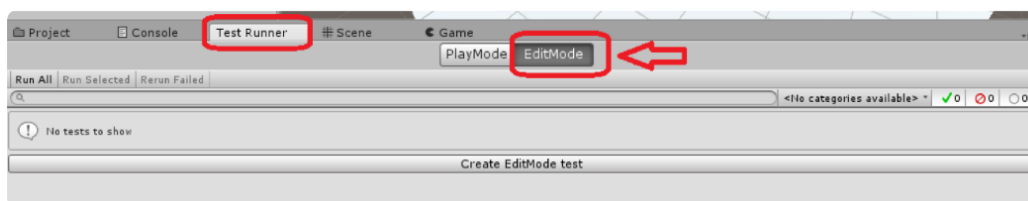
**Figura 37. Mensaje de error**



## Modo edición

Cuando la verificación se hace utilizando el modo de edición, entonces en la ventana de Test Runner se hace clic en EditMode. Como se mencionó anteriormente este modo es usado cuando se quiere verificar que un elemento realiza adecuadamente un comportamiento que requiere de cálculo y no se ejecuta en tiempo real.

**Figura 38. Modo edición**

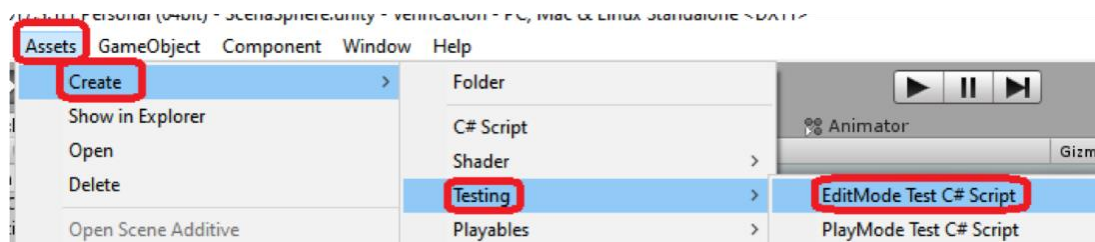


## Script de edición

Una vez activado el modo edición en Test Runner se debe proceder a crear la prueba que se va a utilizar para hacer la verificación del comportamiento de los elementos en la escena:

- Menu -> Assets -> Create -> Testing -> EditMode Test C# Script

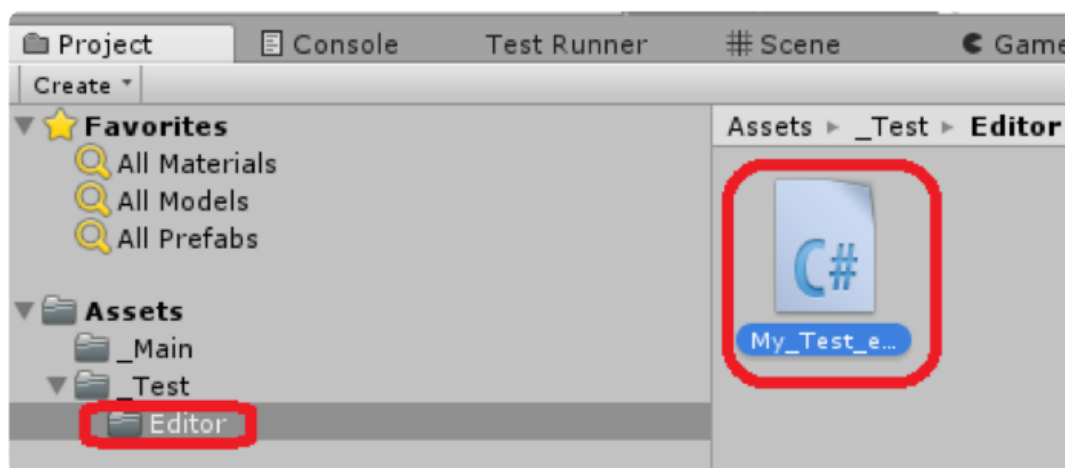
**Figura 39. Script de edición**



### Script My\_Test edición

El Script My\_Test edición que va a servir de testeo se ha creado bajo la carpeta Editor, automáticamente ensamblada por el programa.

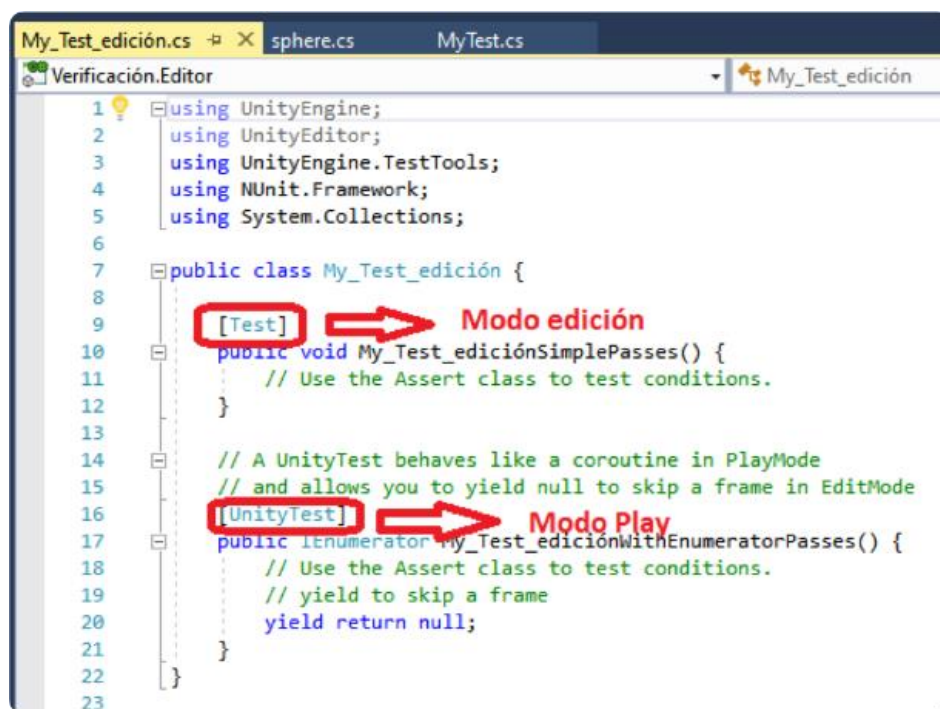
**Figura 40. Script My\_Test edición**



### Clase Test

La diferencia del EditMode con el modo PlayMode, radica en que cuando se escriban las líneas de código en el editor de Visual Studio se deben realizar bajo la clase Test, ya que la clase Unity Test corresponde al modo Play o Run time.

**Figura 41. Clase Test**



```

1  using UnityEngine;
2  using UnityEditor;
3  using UnityEngine.TestTools;
4  using NUnit.Framework;
5  using System.Collections;
6
7  public class My_Test_edición {
8
9      [Test] ➡ Modo edición
10     public void My_Test_ediciónSimplePasses() {
11         // Use the Assert class to test conditions.
12     }
13
14     // A UnityTest behaves like a coroutine in PlayMode
15     // and allows you to yield null to skip a frame in EditMode
16     [UnityTest] ➡ Modo Play
17     public IEnumerator My_Test_ediciónWithEnumeratorPasses() {
18         // Use the Assert class to test conditions.
19         // yield to skip a frame
20         yield return null;
21     }
22 }
23

```

Para profundizar en el tema con algunos consejos sobre cómo usar Unity Test Framework, consultar el tema “Prueba el código de tu juego con Unity Test Framework” que se encuentra dentro del material complementario.

## Síntesis

A continuación, la síntesis del componente formativo abordado:



## Material complementario

Tema	Referencia APA del Material	Tipo de material	Enlace del Recurso o Archivo del documento material
Diseño del ciclo de iteraciones – <i>backlog</i>	Deloitte. (s.f). <i>Artefactos Scrum: las 3 herramientas clave de gestión.</i>	[Página web]	<a href="https://www2.deloitte.com/es/es/pages/technology/articles/artefactos-scrum.html">https://www2.deloitte.com/es/es/pages/technology/articles/artefactos-scrum.html</a>
Diseño del ciclo de iteraciones – <i>backlog</i>	Alicia Raeburn. (2022). <i>Qué es product backlog y guía para hacer uno con ejemplo.</i>	[Página web]	<a href="https://proagilist.es/blog/agilidad-y-gestion-agil/agile-scrum/backlog-de-producto-en-scrum/">https://proagilist.es/blog/agilidad-y-gestion-agil/agile-scrum/backlog-de-producto-en-scrum/</a>
Prototipado digital con paquetes prefabricados	Monteserín, P. (s.f). <i>Instanciar un objeto en Unity 3D.</i>	[Blog]	<a href="https://pablomonteserin.com/curso/unity-3d/instanciar/">https://pablomonteserin.com/curso/unity-3d/instanciar/</a>
Prototipado digital con paquetes prefabricados	Unity. Documentation. (2016). <i>Prefabs.</i>	[Manual web]	<a href="https://docs.unity3d.com/es/530/Manual/Prefabs.html">https://docs.unity3d.com/es/530/Manual/Prefabs.html</a>
Métricas del videojuego a partir de impresiones de detalles de eventos ( <i>debug.log</i> )	Unity. Documentation. (2020). <i>Métricas, segmentos y terminología de Analytics.</i>	[Manual web]	<a href="https://docs.unity3d.com/es/2019.4/Manual/UnityAnalyticsTerminology.html">https://docs.unity3d.com/es/2019.4/Manual/UnityAnalyticsTerminology.html</a>
Métricas del videojuego a partir de	Unity. Documentation. (2021). <i>Depurar.</i>	[Manual web]	<a href="https://docs.unity3d.com/ScriptReference/Debug.html">https://docs.unity3d.com/ScriptReference/Debug.html</a>

impresiones de detalles de eventos ( <i>debug.log</i> )			
Verificación basada en el modelo SCRUM	Trigas, M. (s.f). <i>Metodología Scrum.</i>	[PDF]	<a href="http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf">http://openaccess.uoc.edu/webapps/o2/bitstream/10609/17885/1/mtrigasTFC0612memoria.pdf</a>
Verificación basada en el modelo SUM	SUM. (s.f). <i>SUM para desarrollo de videojuegos.</i>	[Página web]	<a href="http://www.gemserk.com/sum/">http://www.gemserk.com/sum/</a>
Verificación y ajustes en entorno de desarrollo (consola, modo edición)	Unity Technologies. (2021). <i>Prueba el código de tu juego con Unity Test Framework.</i>	[Manual web]	<a href="https://unity.com/es/how-to/unity-test-framework-video-game-development">https://unity.com/es/how-to/unity-test-framework-video-game-development</a>



## Glosario

**Backlog:** listado del trabajo pendiente por realizar y ordenado jerárquicamente por prioridad (García, 2019).

**Debug.log:** método que facilita la depuración mientras se está desarrollando el juego (Unity Documentation, 2021).

**Métricas:** mecanismos para evaluar e identificar la jugabilidad de un videojuego para detectar si es o no jugable (Gonzáles & Gutiérrez, s.f).

**SCRUM:** marco de trabajo que sirve para ayudar a generar valor en la solución de problemas complejos a las personas, equipos de trabajo y organizaciones (Schwaber & Sutherland, 2020).

**SUM:** metodología ágil utilizada para el desarrollo de videojuegos de calidad en menor tiempo y a menor costo, con resultados fácilmente predecibles (Acerenza, et al, 2009).

**Verificación:** proceso de análisis y pruebas en cada fase del desarrollo del software que consiste en revisión de requisitos, diseño, líneas de código y aplicación de pruebas (Hernández & Pérez, 2017).

## Referencias bibliográficas

- Acerenza, N. Et al. (2009). *Una metodología para desarrollo de videojuegos*. [https://www.fing.edu.uy/sites/default/files/biblio/22811/asse\\_2009\\_16.pdf](https://www.fing.edu.uy/sites/default/files/biblio/22811/asse_2009_16.pdf)
- García, R. (2019). *¿Qué es el Backlog?* <https://muyagile.com/que-es-el-backlog/>
- Gonzáles, J. & Gutiérrez, F. (s.f.). *Jugabilidad como medida de calidad en el desarrollo de videojuegos*. [http://ceur-ws.org/Vol-1196/cosecivi14\\_submission\\_23.pdf](http://ceur-ws.org/Vol-1196/cosecivi14_submission_23.pdf)
- Hernández, A. & Pérez, K. (2017). *Criterios para verificar y validar mecanismos en el desarrollo de videojuegos*. <https://zenodo.org/record/2617300/files/n12a1.pdf>
- Schwaber, K. & Stherland, J. (2020). *La guía de Scrum*. <https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf>
- Unity Documentation. (2021). *Depurar*. <https://docs.unity3d.com/ScriptReference/Debug.html>.

## Créditos

Nombre	Cargo	Regional y Centro de Formación
Claudia Patricia Aristizábal Gutiérrez	Responsable del equipo	Dirección General
Liliana Victoria Morales Gualdrón	Responsable de línea de producción	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Olga Lucía Mogollón Carvajal	Experta temática	Centro para la Industria de la Comunicación Gráfica - Regional Distrito Capital
Luz Áida Quintero Velásquez	Diseñadora instruccional	Centro de Gestión Industrial - Regional Distrito Capital
Silvia Milena Sequeda Cárdenas	Evaluadora instruccional	Centro de Gestión Industrial - Regional Distrito Capital
Rafael Neftalí Lizcano Reyes	Asesor pedagógico Ecosistema RED	Centro Industrial del Diseño y la Manufactura - Regional Santander
Julia Isabel Roberto	Diseñadora y evaluador instruccional	Centro para la Industria de la Comunicación Gráfica - Regional Distrito Capital
Gloria Amparo López Escudero	Adecuadora instruccional - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Andrés Felipe Velandia Espitia	Metodólogo para la formación virtual - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Yuly Andrea Rey Quíñonez	Diseñador de contenidos digitales - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Laura Gisselle Murcia Pardo	Animador y productor multimedia - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Jhon Jairo Urueta Alvarez	Desarrollador full-stack - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital

Carolina Coca Salazar	Evaluadora para contenidos inclusivos y accesibles - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Lina Marcela Perez Manchego	Validadora de recursos digitales - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Leyson Fabián Castaño Pérez	Validador de recursos digitales - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital