Características básicas de Unity

Unity permite creación de juegos 2D y 3D. Visualizaciones arquitectónicas o animaciones 3D.
Unity es un motro gráfico similar a *Unreal* pero tiene una curva de aprendizaje más fácil, basado en el lenguaje de programación C# (más práctico que C++ de *Unreal*). En contrapartida, *Unity* para grandes juegos en 3D es algo lento.
Descarga e instalación: Desde *unity3d.com/es* o *unity3d.com/es/get-unity/download*.
Coste: Instala la versión gratuita personal si vas a ganar menos de 1500 €.
Interfaz gráfica:

- *Main Editor Window*: Es la ventana del editor principal. Contiene varias vistas (views)
- *Project View*: Panel de propiedades. Controlas los recursos del proyecto.
- *Hierarchy*: Panel que contiene los objetos de la escena. *GameObject*
- Scene View: ventana interactiva donde seleccionar y posicionar el jugador, la cámara, los enemigos, y otros *GameObjects*.
- *Game View*: Vista desde la cámara representando el producto final.
- *Inspector*: Panel con información detallada del objeto seleccionado *(GameObject).*

Scripting: define el comportamiento del juego.
Funciones básicas: Cuando se crea un nuevo Script, por defecto éste contiene la función Update(), unque tenemos otras opciones / eventos comunes disponibles, como:

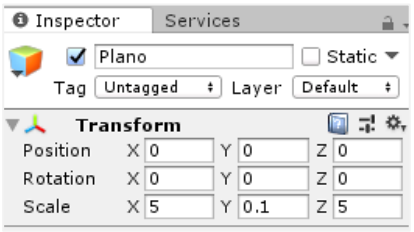
- *FixedUpdate*(): se ejecuta en intervalos regulares.
- *Awake*(): El código dentro de esta función es llamado cuando se inicia el script.
- *Start*(): Se llama antes de cualquier función Update, pero después de la función Awake, si el script es habilitado .
- *OnCollisionEnter*(): Cuando el objeto de juego colisiona con otro objeto de juego.
- *OnMouseDown*(): Cuando el ratón hace clic sobre un objeto de juego que contiene un 'GUIElement' o un 'Collider'.
- *OnMouseOver*(): Cuando el ratón se coloca sobre un objeto de juego que contiene un 'GUIElement' o un 'Collider'.

Ejercicio 1. Mover un objeto en el juego.

- Creamos un nuevo proyecto: ***+ New*** *Proje*ct o del menú: *File – New Project*.
  Seleccionamos la localización del proyecto y pulsamos en Create project. La nombramos "***ejercicio1***".
- Creamos una escena. *File->New Scene*. La nombramos "***escena1***

Entorno del jugador:
Para situar la escena, crearemos una superficie en la que camine el usuario con forma de cubo.

- Crea un cubo (Create – 3DObject – Cube)
- Desde el *inspector*, *escala* x, y, z a 5, 0.1, y 5 respectivamente, para ampliar y aplastar el cubo. (Equivaldría a un suelo de 5x5 metros)
- Desde el inspector, cambia el cuadro del nombre del objeto por: *Plano*.
- Crea una esfera (Create – 3DObject – Sphere) y colócala en el centro del plano. Si no puedes ver los objetos desde la Vista de Juego (***Game View***), cambia la cámara principal para que esté todo visible. Renombra elobjeto como *Cubo1*.



- Orbita y encuadra, manteniendo el botón derecho del mouse.
- También deberías crear un punto de luz (Create – Light – Point light) y colocarlo encima de los objetos de forma que sean visibles más fácilmente. (Tirando del gizmo verde vertical hacia arriba).

Escribir el Script
Vamos a mover la esfera. Para esto vamos a escribir un script que leerá la dirección desde el teclado y luego adjuntamos (asociamos) el script a la esfera.

- Comenzar creando un script vacío. Elije: ***Assets->Create->C#Script*** (Si instalaste el complemento de *Microsoft Visual Studio C#)* y renombra este script como *Movimiento1* en el *Panel Project.*
  Otra opción es crear el componente sobre el objeto: Con la esfera seleccionada, desde el inspector escoger: ***Add component*** – ***New script*** *Movimiento1.*
- Si instalaste el complemento de *Microsoft Visual Studio C# script*, haz *doble click* sobre el Script *C#.* Se abrirá en C# con las funciones *Start()* y **`Update()`** ya insertadas. Utilizaremos la función ***Update*** que es el comportamiento por defecto para cualquier código que queremos que se ejecute a cada frame del juego.
- Para mover un objeto necesitamos cambiar la propiedad de *posición* de su *transform*, la función **`Translate`** perteneciente al transform nos permitirá hacerlo. La función Translate necesita 3 parámetros: los movimientos x, y, z. Si queremos controlar el objeto con las teclas del cursor, simplemente asociamos código para determinar si las teclas del cursor están siendo presionadas para los respectivos parámetros:

```
void Update () {
  transform.Translate(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
}
```

La función **`Input.GetAxis()`** devuelve un valor entre -1 y 1. Por ejemplo, en el eje horizontal, la tecla izquierda del cursor devuelve -1, la tecla derecha del cursor devuelve 1.

En Unity, el eje "Y" es el vertical, de ahí el parámetro 0 en el eje 'Y' ya que no estamos interesados en mover la cámara hacia arriba. Los ejes Horizontal y Vertical están predefinidos en el **Input Settings.** Estos nombres pueden ser fácilmente cambiados en **Edit->Project Settings->Input**.
-Abre el C#Script *Movimiento1* y escribe el código superior, presta atención a las mayúsculas.

Adjuntar o asociar el script al objeto:
Si no has creado el script en el objeto, debes asociarselo: haz click en la esfera desde la *Vista de Jerarquía* (**Hierarchy View)** o desde la *Vista de Escena* (**Scene View**).
Con la esfera seleccionada, elige del menú: **Components->Scripts**: Movimiento1. En la Vista de Inspector (**Inspector View**), verás que aparece el apartado: ☑ **Movimiento 1 (script).**
Truco: puedes también asignar un script a un objeto arrastrando el script desde la Vista **Project** sobre el objeto en la **View Scene**.
-Pon en marcha el juego (presiona el icono 'play' en la parte superior), deberías ser capaz de mover la esfera con las teclas del cursor o W, S, A, D.
Como probablemente, la esfera se mueva demasiado rápido, vamos a buscar una mejor forma de controlar su velocidad.

Como el anterior código estaba dentro de la función Update(), la esfera se movía a la velocidad medida en metros por frame. De todas formas, es mejor asegurarse de que tus objetos de juego se mueven al ritmo predecible de metros por segundo. Para conseguir esto tenemos que multiplicar el valor dado por la función `Input.GetAxis()` por el *tiempo Delta* y también por la velocidad a la que queremos movernos por segundo:

```
float speed = 5.0f;
void Update() {
        float x = Input.GetAxis("Horizontal") * Time.deltaTime * speed;
        float z = Input.GetAxis("Vertical") * Time.deltaTime * speed;
        transform.Translate(x, 0, z);
    }
```

-Actualiza el script Movimiento1 con el código superior.

Date cuenta aquí que la velocidad variable se declara fuera de la función Update(), esto es llamado una variable expuesta (exponed variable) o pública, y aparecerá en el `Inspector View` para cualquier objeto de juego al que esté adjunto el script
Exponer variables es útil cuando el valor necesita ser modificado para conseguir el efecto deseado, esto es mucho más fácil que cambiar códigos.

Conectar variables

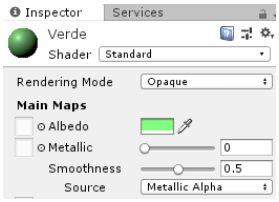
Conectar variables a través de GUI es una característica muy poderosa de Unity. Permite que las variables que normalmente estarían asignadas en código puedan ser hechas mediante el drag and drop (arrastrar y tirar) en la GUI de Unity.
Necesitaremos exponer una variable en nuestro código de script para poder asignar el parámetro en el **Inspector View**.
Vamos a demostrar el concepto de conectar variables creando un foco de luz que seguirá al objeto mientras se mueve.
Añade un foco de luz (**spotlight**) a la Vista de Escena (**Scene View**). Muévelo si es necesario para que esté cerca de los otros objetos de juego.
Crea un nuevo Script y nómbralo 'Follow' (Seguir).
Para que nuestro nuevo foco de luz apunte a un objeto, hay una función en Unity que lo hace: ***transform.LookAt().***
Ahora vamos a la sección de conectar variables; ¿qué usamos como un parámetro para **LookAt()**? Podemos insertar un objeto de juego, no obstante, sabemos que queremos asignar la variable mediante la GUI, así que sólo tendremos que usar una variable expuesta (del tipo **Transform**). Nuestro **Follow.js** script debería parecerse a esto:

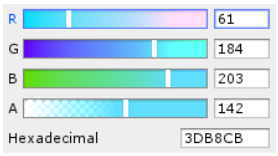
```
Transform target; //variable expuesta del tipo Transform;
function Update () {
  transform.LookAt(target);
}
```

Adjunta el script al foco de luz y fijate que cuando el componente se añade, la variable "**target**" (objetivo) está expuesta.
Con el foco de luz aun seleccionado, arrastra la esfera desde la Vista de Jerarquía (**Hierarchy View**) hasta la variable "**target**" en la Vista de Inspector (**Inspector View**).
Pon en marcha el juego. Si miras la Vista de Escena (**Scene View**) deberías ver el foco de luz siguiendo a la esfera. Puedes que quieras cambiar la posición del foco para mejorar el efecto.
Guarda el proyecto: *File – Save Project*.

Crear un juego de estilo Breakout.

- Crear un cubo: (*Create – 3DObject – Cube*)
- Crear 1 cubo alragado para la pared izquierda (Pos: -10,0,0 Scale: 1,20,1)
- Duplicar (*Edit – Duplicate*) y deplazar a la posición: 10,0,0 para la pared derecha. Nombre: pared2
- Duplicar y rotar el cubo sobre el eje z. Nombrar: techo.
  (Position: 0,10,0, Rotation: 0,0,90, Scale: 1,24,1)
- Crear los ladrillos (17) duplicando por columnas
  Puedes arrastrar todos los ladrillos sobre el primero para agruparlos desde el explorador jerárquico *Hierachy*.
- Crear la bola (*Create – 3d object – Sphere*)
- Crear un cubo para el suelo. (Pos: 0, -22, 0 Scale: 50, 20, 50)
- Crear una carpeta en Assets llamada materiales (*Assets – Create – Folder*)
  Dentro de la carpeta, crear 3 materiales (Assets - *Create – Material*) llamados: Rojo, suelo y verde. Escoger el color desde el inspector: Rojo, azul y verde. Asignar cada material al objeto correspondiente, arrastrando el material encima del objeto.



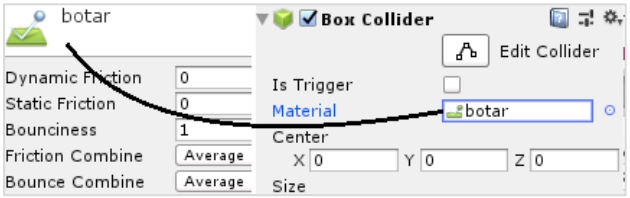


  Al suelo le podemos dar una sensación de azul líquido variando el valor alfa de transparencia

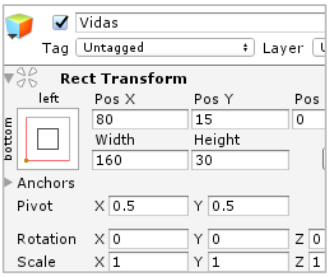
- Seleccionar la pala y añadir el componente *RigidBody* (Component – Physics – Rigibody) para convertirla en cuerpo sólido o cuerporígido. Añadir también este componente a la Bola.
- Añadir el Script a la pala, de movimiento con las flechas: *Assets – Create – C#Script:*

```
public class Pala : MonoBehaviour {
    public float velocipala = 1f;  //variable pública para la velocidad de la pala del tipo numérico decimal
    void Update () {        // es llamado en cada impulso
        float Posx = transform.position.x + (Input.GetAxis("Horizontal") * velocipala); //coge variabe Posx
        transform.position = new Vector3(Posx, -9f, 0f); //movemos la pala a la nueva posición del objeto
    }
```

  Arrastrar este Script sobre la pala. Al correr el juego, debería de moverse la pala hacia los lados.
- Añadir efecto físico de rebote*:*
  En la carpeta de *Materiales* añadir material físico (Create – Physics material) Nombre: botar. Establecer los valores de la imagen.
  Añadir a la pala el efecto de colisión: (Component – Physis – Box collider) y arrastar botar hasta Material
  Añadir el mismo efecto de colisión a las dos paredes, el techo y a todos los ladrillos.
  Añadir a la bola el efecto de colisión esférica: (Component – Physis – Sphere collider) y arrastar *botar* hasta *Material*. Radius: 0.5



- Añadir texto (*GameObject – UI – Text*): Aparecerá en canvas con un eventsystem
  Para la puntuación. Nombre: Vidas texto: Lives: 3. Poner tamaño/ posición de la imagen →
  Añadir otro texto. Nombre: GameOver texto: GAME OVER fuente: Arial 40 bold negrita
  Desactiva la casilla junto al nombre ✓ para que no se muestre al principio.
  Añadir otro texto. Nombre: YouWon texto: GANASTE - YOU WON !! desactiva disable ✓



- Añadir el efecto de sistema de partículas para la caída (*GameObject – Effects – Particle Systems*)
  Nombre: *ParticulasBrick.* Rotation x: 90 (hacia abajo). sin ✓ loop, duración: 1
  Activar: rotación y tamaño mientras dure: ✓ Rotation over lifetime ✓ Size over lifetime
  Puedes añadir un nuevo material en la carpeta de materiales (*Create – Material*) para asignar a las particulas el color rojo oscuro.

- Añadir el código:

Extracto de: unity3d.com/es/learn/tutorials/modules/beginner/live-training-archive/creating-a-breakout-game?playlist=41639

Uses: using UnityEngine;

```
public class Pala : MonoBehaviour {
  public float velocipala = 1f;
  private Vector3 posicionjugador;
  void Update () {
  float Posx = transform.position.x +
(Input.GetAxis("Horizontal") * velocipala);
  posicionjugador = new Vector3(Posx, -9.5f, 0f);
  transform.position = posicionjugador;
  }
}

public class Bola : MonoBehaviour {
  public float velocidadinicial = 600f;
  private Rigidbody rb;
  private bool ingame;
  void Awake () {// Use this for initialization
    rb = GetComponent<Rigidbody>();
    }
    void Update () {
    if (Input.GetButtonDown("Fire1") && ingame ==
false)
        {
        transform.parent = null;
        ingame = true;
        rb.isKinematic = false;
        rb.AddForce(new Vector3(velocidadinicial,
velocidadinicial, 0));
        }
    }
}

public class Bricks : MonoBehaviour
{
    public GameObject brickParticle;
    void OnCollisionEnter(Collision other)
    {
     Instantiate(brickParticle, transform.position,
Quaternion.identity);
     GM.instance.DestroyBrick();
     Destroy(gameObject);
    }
}

public class DeadZone : MonoBehaviour
{
    void OnTriggerEnter(Collider col)
    {
        GM.instance.LoseLife();
    }
}
```

```
public class GM : MonoBehaviour
{
    public int lives = 3;
    public int bricks = 18;
    public float resetDelay = 1f;
    public Text livesText;
    public GameObject gameOver;
    public GameObject youWon;
    public GameObject bricksPrefab;
    public GameObject paddle;
    public GameObject deathParticles;
    public static GM instance = null;
    private GameObject clonePaddle;
    void Awake()    // Use this for initialization
    {
        if (instance == null)
            instance = this;
        else if (instance != this)
            Destroy(gameObject);
        Setup();
    }
    public void Setup()
    {
     clonePaddle = Instantiate(paddle,
transform.position, Quaternion.identity) as
GameObject;
     Instantiate(bricksPrefab, transform.position,
Quaternion.identity);
    }

 void CheckGameOver()
    {
        if (bricks < 1)
        {
           youWon.SetActive(true);
           Time.timeScale = .25f;
           Invoke("Reset", resetDelay);
        }

        if (lives < 1)
        {
           gameOver.SetActive(true);
           Time.timeScale = .25f;
           Invoke("Reset", resetDelay);
        }
    }
    void Reset()
    {
     Time.timeScale = 1f;
     Application.LoadLevel(Application.loadedLevel);
    }

    public void LoseLife()
    {
     lives--;
     livesText.text = "Lives: " + lives;
     Instantiate(deathParticles,
clonePaddle.transform.position, Quaternion.identity);
     Destroy(clonePaddle);
     Invoke("SetupPaddle", resetDelay);
     CheckGameOver();
    }
    void SetupPaddle()
    {
    clonePaddle = Instantiate(paddle,
transform.position, Quaternion.identity) as
GameObject;
    }
    public void DestroyBrick()
    {
        bricks--;
        CheckGameOver();
    }
}
```