



Universidad Tecnológica  
del Norte de Guanajuato  
Organismo Público Descentralizado del Gobierno del Estado  
“Educación y progreso para la vida”

## INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN ÁREA ENTORNOS VIRTUALES Y NEGOCIOS DIGITALES.

“Programación de videojuegos 1 unidad II”

Tutorial 5:

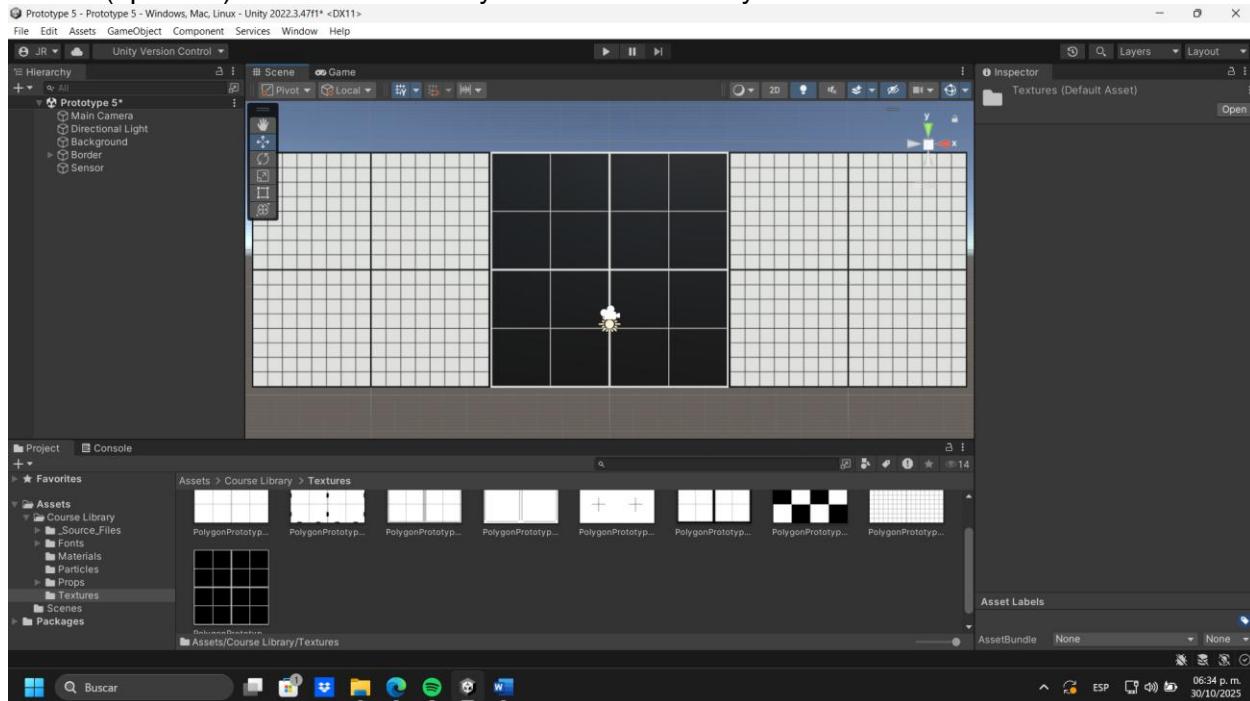
# Documentación del tutorial 5

Profesor: Gabriel Barrón Rodríguez.

Alumno: Jesús Alberto Arriaga Ramírez  
No. de Control: 1223100850

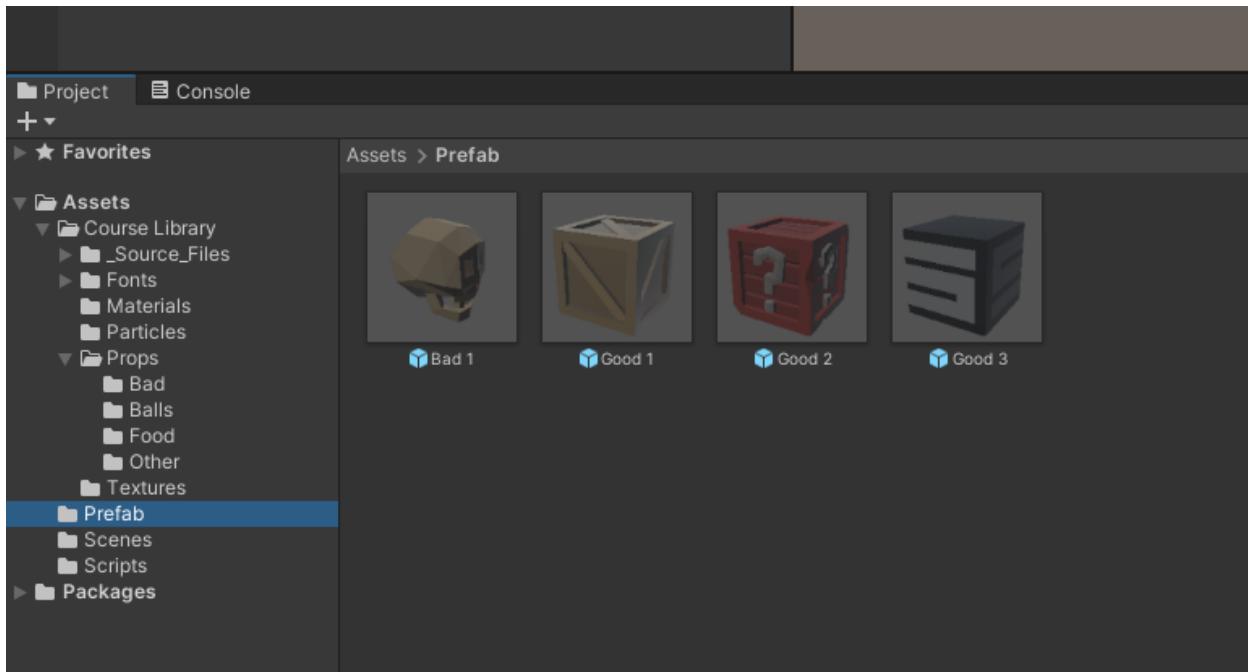
30 de octubre del 2025.

1. Abre **Unity Hub** y crea un proyecto «Prototype 5» vacío en el directorio de tu curso en la versión correcta de Unity. Si olvidaste cómo hacerlo, consulta las instrucciones en la [Lección 1.1. Paso 1.](#)
2. Haz clic para descargar los [Archivos de inicio del Prototype 5](#), **extrae** la carpeta comprimida y después **importa** el .unitypackage a tu proyecto. Si olvidaste cómo hacerlo, consulta las instrucciones en la [Lección 1.1. Paso 2.](#)
3. Abre la Escena **Prototype 5** y elimina la **Escena de muestra** sin guardar.
4. Haz clic en el **ícono 2D** en la vista de Escena para cambiar la vista de la Escena a **2D**.
5. (opcional) Cambia la textura y el color del **fondo** y el color de los **bordes**.



*Lo primero que necesitamos en nuestro juego son tres objetos buenos para recolectar y un objeto malo para evitar. Te corresponde a ti decidir qué es bueno y qué es malo.*

1. De **Library**, arrastra 3 objetos «buenos» y 1 objeto «malo» hasta la Escena, cambia sus nombres a [«Good 1»](#), [«Good 2»](#), [«Good 3»](#) y [«Bad 1»](#).
2. Agrega los componentes **Rigidbody** y **Box Collider**, después asegúrate de que los colisionadores rodeen a los objetos de forma adecuada.
3. Crea una nueva carpeta **Scripts** con un nuevo Script [«Target.cs»](#) dentro y adjúntala a los **objetivos**.
4. Arrastra los 4 objetivos a la **carpeta Prefabs** para crear «original Prefabs», luego **eliminalos** de la Escena.



Ahora que tenemos 4 Prefabs objetivo con el mismo script necesitamos lanzarlos por el aire con fuerza, torsión y posición aleatorias.

1. En **Target.cs**, declara un nuevo **private Rigidbody targetRb;** e inicialízalo en **Start()**.
2. En **Start()**, agrega un **impulso hacia arriba** multiplicado por una **velocidad aleatoria**.
3. Agrega una **torsión** con valores **XYZ aleatorios**.
4. Define la **posición** con un **valor X aleatorio**.

```

    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;

    public class Target : MonoBehaviour
    {
        private Rigidbody targetRb;

        void Start()
        {
            targetRb = GetComponent<Rigidbody>();

            targetRb.AddForce(Vector3.up * Random.Range(12, 16), ForceMode.Impulse);
            targetRb.AddTorque(Random.Range(-10, 10), Random.Range(-10, 10), Random.Range(-10, 10), ForceMode.Impulse);

            transform.position = new Vector3(Random.Range(4, 4), -6);
        }

        void Update()
        {
        }
    }

```

En lugar de dejar las fuerzas, torsiones y posiciones aleatorias que hacen que nuestra función **Start()** sea complicada e ilegible, vamos a almacenar cada una de ellas en métodos personalizados nuevos y les daremos nombres claros.

1. Declara e inicializa nuevas variables private float para ***minSpeed***, ***maxSpeed***, ***maxTorque***, ***xRange***, y ***ySpawnPos***;
2. Crea una nueva función para ***Vector3 RandomForce()*** y llámala en ***Start()***
3. Crea una nueva función para ***float RandomTorque()*** y llámala en ***Start()***
4. Crea una nueva función para ***RandomSpawnPos()*** haz que devuelva un nuevo ***Vector3*** y llámala en ***Start()***

```

void Start()
{
    targeRb = GetComponent<Rigidbody>();

    targeRb.AddForce(RandomForce(), ForceMode.Impulse);
    targeRb.AddTorque(RandomTorque(), RandomTorque(), RandomTorque(), ForceMode.Impulse);

    transform.position = RandomSpawnPos();
}

// Update is called once per frame

void Update()
{
}

Vector3 RandomForce()
{
    return Vector3.up * Random.Range(minSpeed, maxSpeed);
}

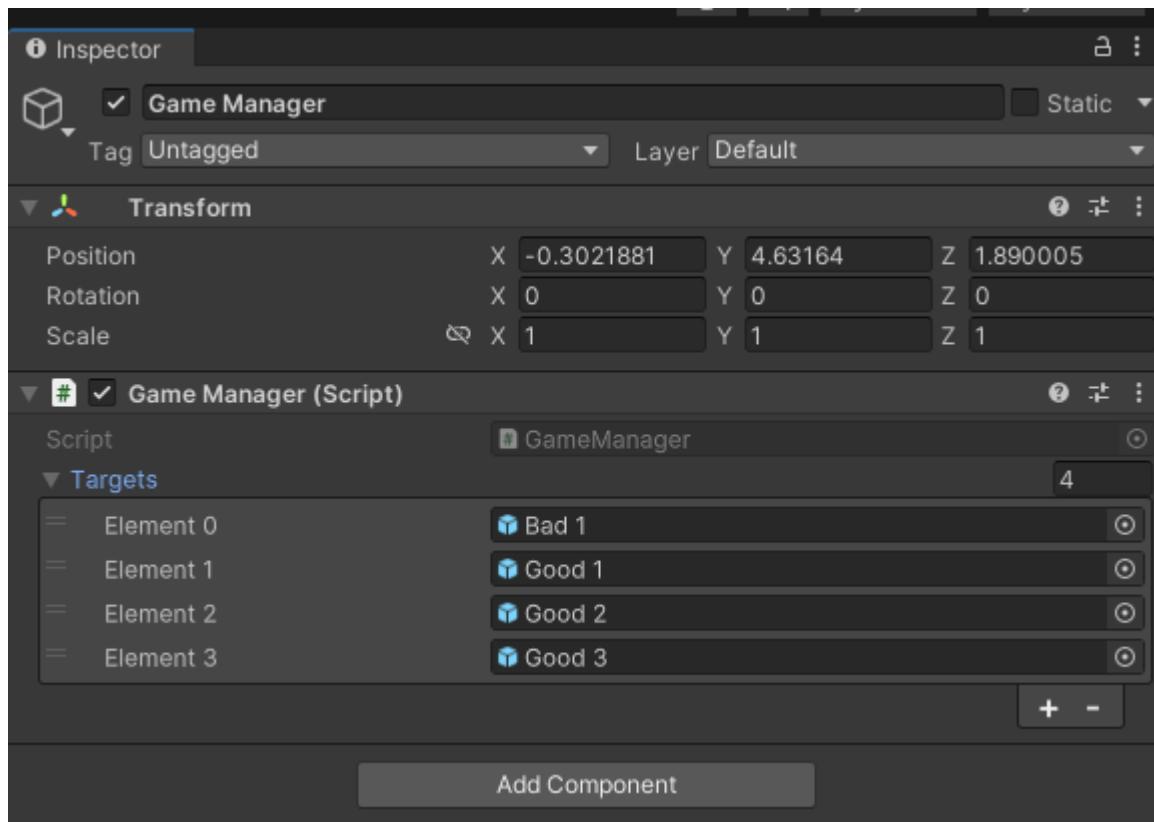
3 referencias
float RandomTorque()
{
    return Random.Range(-maxTorque, maxTorque);
}

1 referencia
Vector3 RandomSpawnPos()
{
    return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
}

```

Lo siguiente que debemos hacer es crear una lista desde la cual se generarán estos objetos. En lugar de crear un *Spawn Manager* para estas funciones de *instanciación*, vamos a crear un *Game Manager* (*Administrador del juego*) que también controlará los estados del juego más adelante.

1. Crea un nuevo objeto vacío «Game Manager», adjunta un nuevo Script ***GameManager.cs*** y ábrelo.
2. Declara una nueva ***public List<GameObject> targets;***, después en el Inspector de Game Manager cambia el ***Size*** de la lista a 4 y asigna tus ***Prefabs***.



Ahora que tenemos una lista de Prefabs de objetos, debemos crear instancias de ellos en el juego mediante corutinas y un nuevo tipo de bucle.

1. Declara e inicializa una nueva variable ***private float spawnRate***.
2. Crea un nuevo método ***IEnumerator SpawnTarget()***.
3. Dentro del nuevo método, ***while(true)***, espera 1 segundo, genera un índice aleatorio, y genera un **objetivo aleatorio**.
4. En ***Start()***, utiliza el método ***StartCoroutine*** para comenzar a generar objetos.

```

public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(SpawnTarget());
    }

    // Update is called once per frame
    void Update()
    {

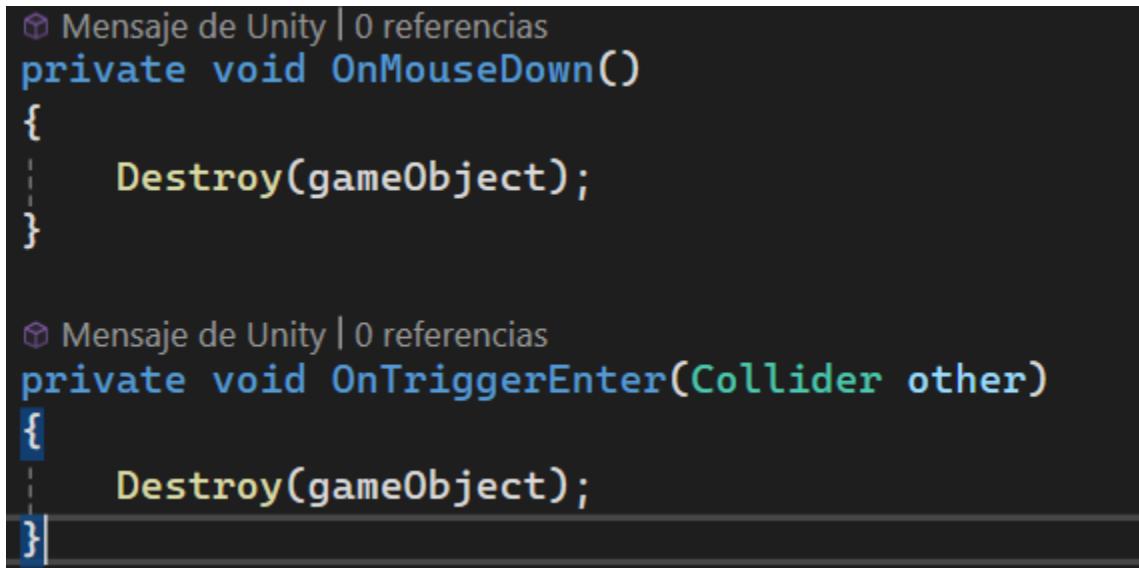
    }

    1 referencia
    IEnumerator SpawnTarget()
    {
        while (true)
        {
            yield return new WaitForSeconds(spawnRate);
            int index = Random.Range(0, targets.Count);
            Instantiate(targets[index]);
        }
    }
}

```

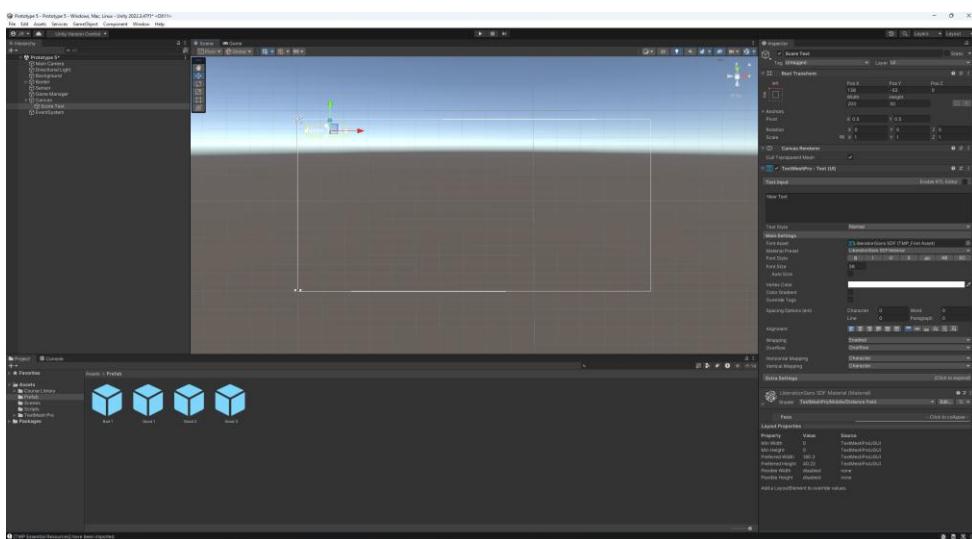
Ya que nuestros objetivos se generan y lanzan al aire, necesitamos una forma para que el jugador pueda destruirlos con un clic. También necesitamos destruir cualquier objetivo que caiga y salga por la parte inferior de la pantalla.

1. En **Target.cs**, agrega un nuevo método para **private void OnMouseDown() {}** y dentro de ese método, destruye el gameObject.
2. Agrega un nuevo método para **private void OnTriggerEnter(Collider other)** y dentro de esa función, destruye el gameObject.



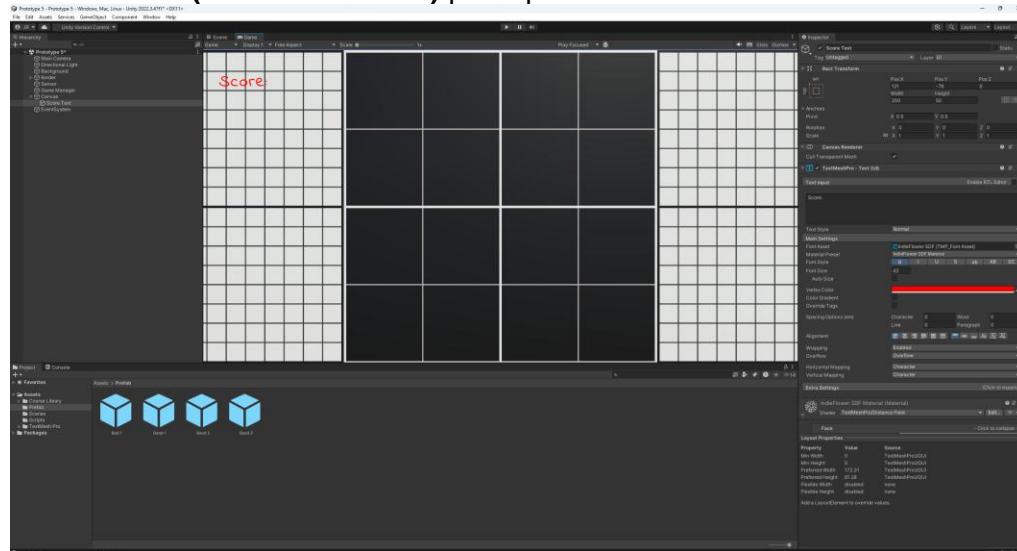
Para poder mostrar el puntaje en la pantalla, necesitamos agregar nuestro primer elemento de la interfaz de usuario.

1. En la Hierarchy, **Create > UI > TextMeshPro text**, después, si se te solicita, haz clic en el botón para **Import TMP Essentials**.
2. Cambia el nombre del nuevo objeto «Score Text» (Texto del puntaje), después **alejate** para ver el **canvas** en la vista de Escena.
3. Cambia el **Anchor Point** para que esté anclado desde la esquina **superior izquierda**.
4. En el Inspector, cambia su **Pos X** y **Pos Y** para que esté en la esquina superior izquierda.



Ahora que el texto básico está en la Escena y en la posición correcta, debemos editar sus propiedades para que luzca bien y tenga el texto correcto.

1. Cambia el texto a «Score:» (Puntaje).
2. Elige un **Font Asset (Recurso de fuente)**, **Style (Estilo)**, **Size (Tamaño)** y **Vertex Color (Color del vértice)** para que se vean bien en contraste con el fondo.



Tenemos un excelente lugar para mostrar el puntaje en la interfaz de usuario, pero no se ve nada ahí. Necesitamos que la interfaz de usuario muestre una variable de puntaje para que el jugador conozca sus puntos.

1. En la parte superior de **GameManager.cs**, agrega «using TMPro;».
2. Declara un nuevo **public TextMeshProUGUI scoreText**, después asigna esa variable en el Inspector.
3. Crea una nueva variable **private int score** e inicialízala en **Start()** como **score = 0;**
4. También en **Start()**, define **scoreText.text = "Score: " + score;**

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    public TextMeshProUGUI scoreText;
    private int score;
    private float spawnRate = 1.0f;

    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(SpawnTarget());
        score = 0;
        scoreText.text = "Score: " + score;
    }

    // Update is called once per frame
    void Update()
    {
    }

    IEnumerator SpawnTarget()
    {
        while (true)
        {
            yield return new WaitForSeconds(spawnRate);
            int index = Random.Range(0, targets.Count);
            Instantiate(targets[index]);
        }
    }
}

```

El texto de puntaje muestra la variable de puntos a la perfección, pero nunca se actualiza. Necesitamos escribir una nueva función que acumule los puntos para mostrarlos

en la interfaz de usuario.

1. Crea un nuevo método **private void UpdateScore** que requiera un parámetro **int scoreToAdd**.
2. Corta y pega **scoreText.text = "Score: " + score;** en el nuevo método, después llama al **UpdateScore(0)** en **Start()**.
3. En **UpdateScore()**, incrementa el puntaje al sumar **score += scoreToAdd;**.
4. Llama al **UpdateScore(5)** en la función **spawnTarget()**.

```
1 referencia
IEnumerator SpawnTarget()
{
    while (true)
    {
        yield return new WaitForSeconds(spawnRate);
        int index = Random.Range(0, targets.Count);
        Instantiate(targets[index]);
        UpdateScore(5);
    }
}

2 referencias
void UpdateScore(int scoreToAdd)
{
    score += scoreToAdd;
    scoreText.text = "Score: " + score;
}
```

Ahora que tenemos un método para actualizar el puntaje, debemos llamar al Script de los objetivos cada vez que uno se destruya.

1. En GameManager.cs, haz que el método **UpdateScore** sea **public**.
2. En Target.cs, crea una referencia a **private GameManager gameManager;**.
3. Inicializa GameManager en **Start()** con el método **Find()**.
4. Cuando un objetivo se destruya, llama a **UpdateScore(5)**; después elimina la llamada del método en **SpawnTarget()**.

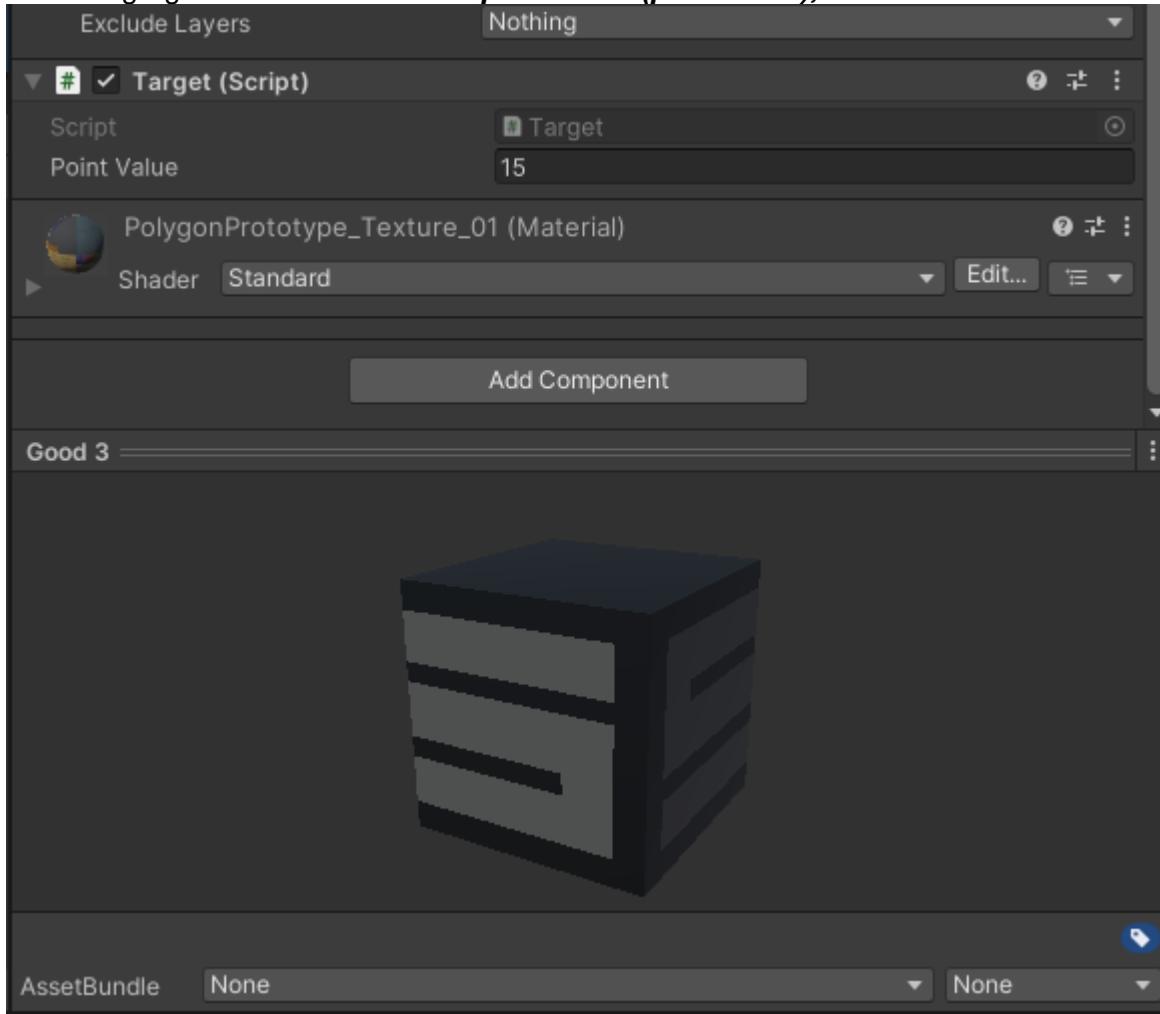
```
}
private void OnMouseDown()
{
    Destroy(gameObject);
    gameManager.UpdateScore(5);
}
```

El puntaje se actualiza cuando se hace clic en los objetivos, pero queremos darle a cada uno de los objetivos un valor distinto. Los objetivos buenos deben tener un puntaje variado y los objetivos malos deben restar puntos.

1. En Target.cs, crea una nueva variable **public int pointValue**.
2. En cada uno de los Inspectors del **Target Prefab** define **Point Value** al valor que

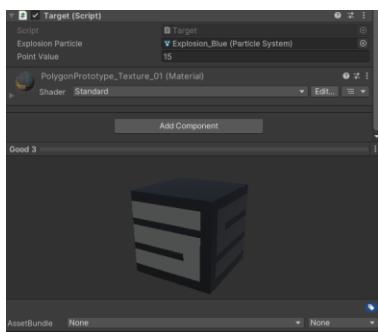
definas para cada uno de ellos, incluido el **valor negativo del objetivo malo**.

3. Agrega la nueva variable a **UpdateScore(pointValue);**



*El puntaje es totalmente funcional, pero hacer clic en los objetivos es un poco... insatisfactorio. Para darle un poco más de interés, agreguemos algunas partículas explosivas cada vez que se hace clic en un objetivo.*

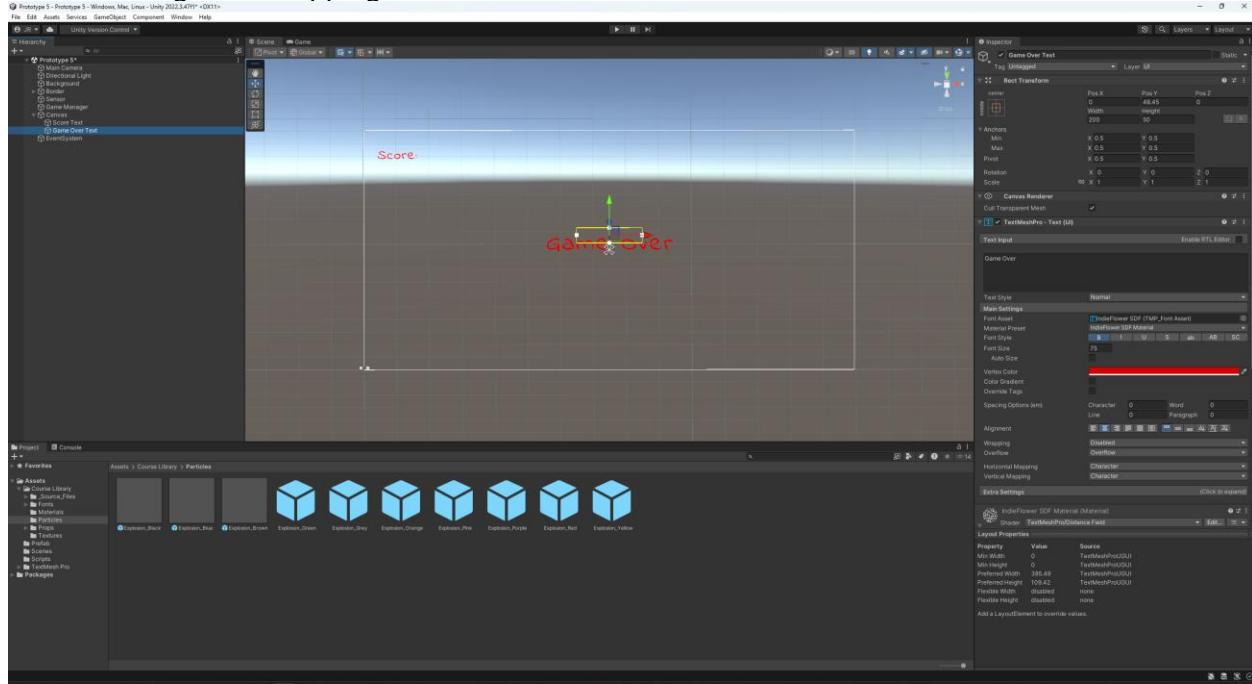
1. En Target.cs, agrega una nueva variable **public ParticleSystem explosionParticle**.
2. Por cada uno de tus Prefabs de objetivo, asigna **Prefab de partículas** en Course Library > Particles a la variable **Explosion Particle**.
3. En la función **OnMouseDown()**, crea una instancia de un nuevo Prefab de explosión.



*Si queremos que aparezca texto de «Game Over» cuando el juego termine, lo primero que debemos hacer es crear y personalizar un nuevo elemento de texto de interfaz*

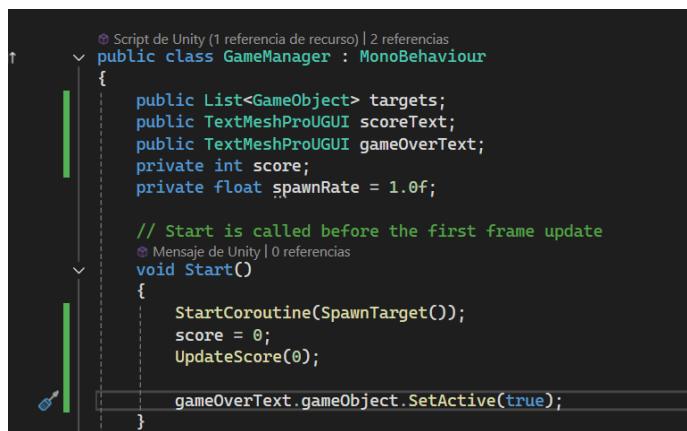
de usuario que diga «Game Over».

1. Haz clic derecho en el **Canvas**, crea un nuevo objeto **UI > TextMeshPro - Text** y cambia su nombre a «Game Over Text».
2. En el Inspector, edita su **Text**, **Pos X**, **Pos Y**, **Font Asset**, **Size**, **Style**, **Color** y **Alignment**.
3. Configura «**Wrapping**» a «**Disabled**».



Tenemos un hermoso texto de fin del juego en la pantalla, pero por el momento solo está ahí y bloquea nuestra visibilidad. Debemos desactivarlo para que pueda reaparecer cuando finalice el juego.

1. En **GameManager.cs**, crea un nuevo **public TextMeshProUGUI gameOverText;** y asignale el objeto **Game Over** en el Inspector.
2. **Desmarca** la casilla de selección Active para **desactivar** el texto de Game Over de forma predeterminada.
3. En **Start()**, activa el texto de Game Over.



Hicimos aparecer el texto «Game Over» al inicio del juego, pero en realidad queremos que lo haga cuando fallemos uno de los objetos «buenos» y caiga.

1. Crea una nueva función **public void GameOver()** y mueve el código que activa el texto de fin del juego adentro.
2. En Target.cs, llama a **gameManager.GameOver()** si un objetivo colisiona con el **sensor**.
3. Agrega una nueva etiqueta «Bad» al **objeto malo**, agrega una condición que solo desencadenará el fin del juego si *no* es un objeto malo.

```

private void OnTriggerEnter(Collider other)
{
    Destroy(gameObject);
    if (!gameObject.CompareTag("Bad"))
    {
        gameManager.GameOver();
    }
}

```

*El mensaje «Game Over» aparece exactamente cuando lo queremos, pero el juego mismo continúa ejecutándose. Para en verdad detener el juego y llamarlo un «Game Over», necesitamos que la generación de objetivos se detenga y no se genere más puntaje para el jugador.*

1. Crea un nuevo **public bool isActive;**
2. Como primera línea en **Start()**, define **isActive = true;** y en **GameOver()**, define **isActive = false;**
3. Para evitar la generación de objetivos, en la corutina **SpawnTarget()**, cambia **while (true)** a **while (isActive)**.
4. Para evitar que el puntaje se siga acumulando, en Target.cs, en la función **OnMouseDown()** agrega la condición **if (gameManager.isActive) {**

```

void Start()
{
    isActive = true;
    score = 0;

    StartCoroutine(SpawnTarget());
    UpdateScore(0);
}

// Update is called once per frame
void Update()
{
}

IEnumerator SpawnTarget()
{
    while (isActive)
    {
        yield return new WaitForSeconds(spawnRate);
        int index = Random.Range(0, targets.Count);
        Instantiate(targets[index]);
    }
}

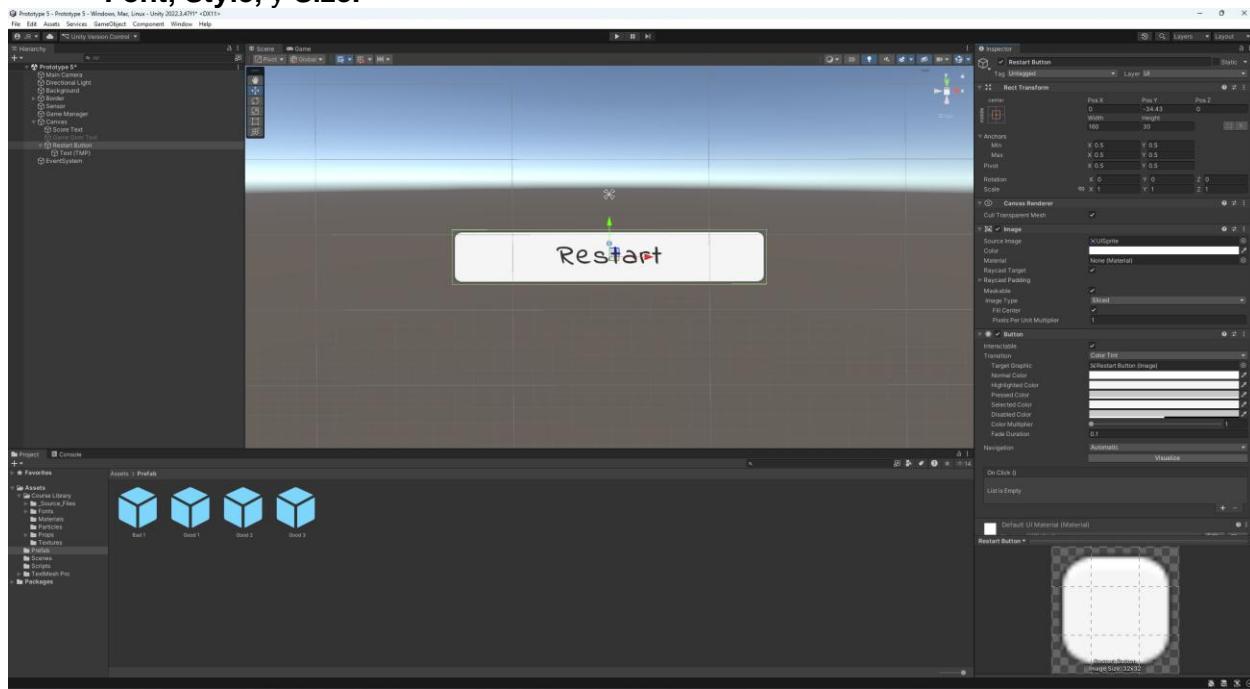
public void UpdateScore(int scoreToAdd)
{
    score += scoreToAdd;
    scoreText.text = "Score: " + score;
}

public void GameOver()
{
    gameOverText.gameObject.SetActive(true);
    isActive = false;
}

```

*Nuestra mecánica de Game Over funciona de maravilla, pero no hay forma de volver a jugar. Para permitir que el jugador reinicie el juego, crearemos nuestro primer botón de la interfaz de usuario.*

1. Haz clic derecho en el **Canvas** y *Create > UI > Button* **Nota:** También podrías usar **Button - TextMeshPro** para tener más control sobre el texto del botón.
2. Cambia el nombre del botón «Restart Button».
3. Por un momento **reactiva** el texto de Game Over para poder reubicar el botón de reinicio correctamente en relación con el texto, después **desactívalo** de nuevo.
4. Selecciona el objeto secundario Text, luego edita su **Text** para que diga «Restart», su **Font**, **Style**, y **Size**.



Agregamos el botón *Restart* a la Escena y se ve GENIAL, pero ahora necesitamos hacer que realmente funcione y reinicie el juego.

1. En **GameManager.cs**, agrega **using UnityEngine.SceneManagement;**.
2. Crea una nueva función **public void RestartGame()** que vuelva a cargar la Escena actual.
3. En el Inspector del **botón** haz clic en **+** para agregar un nuevo **evento On Click**, arrástralos al objeto **Game Manager** y selecciona la función **GameManager.RestartGame**.

```
0 referencias
public void RestartGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

El botón *Restart* se ve muy bien, pero no queremos que nos estorbe durante todo el juego. De manera similar al mensaje de «Game Over», desactivaremos el botón *Restart* mientras el juego esté activo.

1. En la parte superior de GameManager.cs agrega **using UnityEngine.UI;**.
2. Declara un nuevo **public Button restartButton;** y asígnale el **Restart Button** en el Inspector.
3. **Desmarca** la casilla de selección «Active» para el **Restart Button** en el Inspector.
4. En la función **GameOver** activa el **Restart Button**.



```

1 referencia
public void GameOver()
{
    restartButton.gameObject.SetActive(true);
    gameOverText.gameObject.SetActive(true);
    isGameActive = false;
}

0 referencias
public void RestartGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}

```

*Lo primero que debemos hacer es crear todos los elementos de la interfaz de usuario que vamos a necesitar. Esto incluye un título grande y tres botones de dificultad.*

1. Duplica tu **texto Game Over** para crear tu **Title Text**, editar su nombre, texto y todos sus atributos.
2. Duplica tu **Restart Button** y edita sus atributos para crear un botón «Easy».
3. Edita y duplica el nuevo botón Easy para crear un botón «Medium» y un botón «Hard».



*Nuestros botones de dificultad se ven muy bien, pero en realidad no hacen nada. Si van a tener una nueva funcionalidad personalizada, primero necesitamos darles un nuevo Script.*

1. Para los 3 nuevos botones, en el componente Button, en la sección **On Click ()**, haz clic en el botón **menos** (-) para eliminar la funcionalidad `RestartGame`.
2. Crea un nuevo Script **DifficultyButton.cs** y adjúntalo a los **3 botones**.
3. Agrega **`using UnityEngine.UI`** a tus importaciones.
4. Crea una nueva variable **`private Button button;`** e inicialízala en **`Start()`**.

```

    using System.Collections;
    using System.Collections.Generic;
    using UnityEngine;
    using UnityEngine.UI;

    public class DifficultyButton : MonoBehaviour
    {
        private Button button;

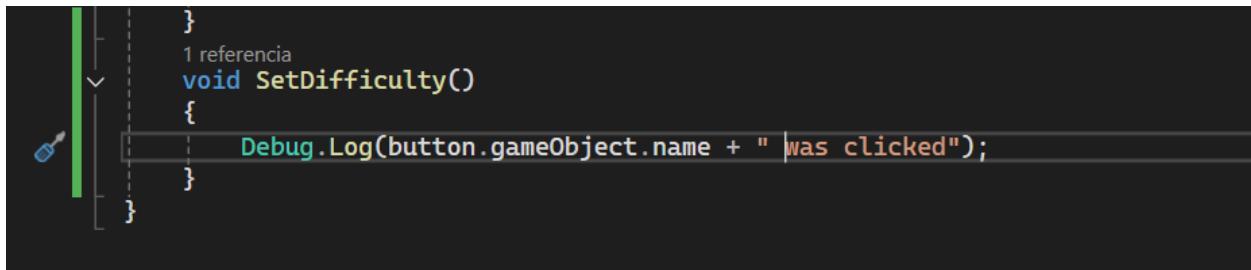
        // Start is called before the first frame update
        void Start()
        {
            button = GetComponent<Button>();
        }

        // Update is called once per frame
        void Update()
        {
        }
    }

```

Ya que tenemos un Script para nuestros botones, podemos crear un método `SetDifficulty` y ligar ese método al clic de dichos botones.

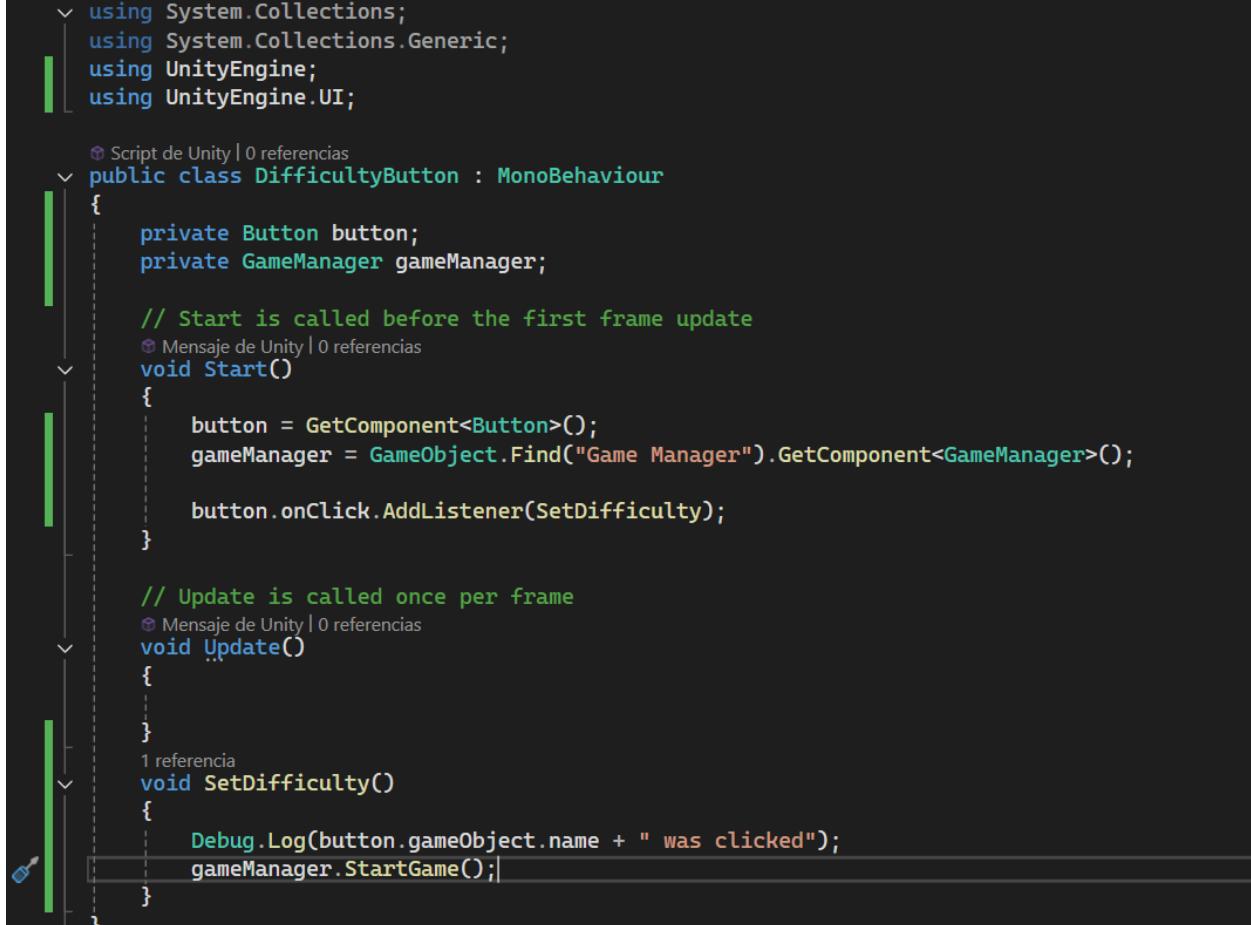
1. Crea una nueva función **`void SetDifficulty`** y dentro **`Debug.Log(gameObject.name + " was clicked");`**.
2. Agrega el **button receptor** para hacer la llamada de la función **`SetDifficulty`**.



```
        }
        1 referencia
    void SetDifficulty()
    {
        Debug.Log(button.gameObject.name + " was clicked");
    }
}
```

La pantalla del título se ve excelente si ignoras los objetivos moviéndose por todos lados, pero no tenemos forma de iniciar el juego en realidad. Necesitamos una función **StartGame** que pueda comunicarse con **SetDifficulty**.

1. En GameManager.cs, crea una nueva función **public void StartGame()** y mueve todo lo que hay en **Start()** ahí.
2. En DifficultyButton.cs, crea un nuevo **private GameManager gameManager;** e inicialízalo en **Start()**.
3. En la función **SetDifficulty()** llama a **gameManager.StartGame();**.



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

// Script de Unity | 0 referencias
public class DifficultyButton : MonoBehaviour
{
    private Button button;
    private GameManager gameManager;

    // Start is called before the first frame update
    // Mensaje de Unity | 0 referencias
    void Start()
    {
        button = GetComponent<Button>();
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();

        button.onClick.AddListener(SetDifficulty);
    }

    // Update is called once per frame
    // Mensaje de Unity | 0 referencias
    void Update()
    {

    }
}

// Update is called once per frame
// Mensaje de Unity | 0 referencias
void SetDifficulty()
{
    Debug.Log(button.gameObject.name + " was clicked");
    gameManager.StartGame();
}
```

Si queremos que la pantalla del título desaparezca cuando el juego comienza, debemos almacenarla en un objeto vacío en lugar de desactivarla de forma individual. Simplemente desactivar un solo objeto vacío primario requiere mucho menos esfuerzo.

1. Haz clic derecho en el Canvas y *Create > Empty Object*, cambia el nombre a «Title Screen» y arrastra los **3 botones** y el **título** ahí.

2. En GameManager.cs, crea un nuevo **public GameObject titleScreen**; y asígnalo en el Inspector.
3. En **StartGame()**, desactiva el objeto de la pantalla de título.

```

0 referencias
public void RestartGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
1 referencia
public void StartGame()
{
    isGameActive = true;
    score = 0;

    StartCoroutine(SpawnTarget());
    UpdateScore(0);

    titleScreen.gameObject.SetActive(false);
}

}

```

*Los botones de dificultad inician el juego, pero todavía no cambian la dificultad del juego. Lo último que tenemos que hacer es lograr que los botones de dificultad realmente afecten la frecuencia con la que se generan los objetivos.*

1. En DifficultyButton.cs, crea una nueva variable **public int difficulty**, después en el Inspector, asigna la dificultad **Easy** como 1, **Medium** como 2 y **Hard** como 3.
2. Agrega un parámetro **int difficulty** a la función **StartGame()**.
3. En **StartGame()**, define **spawnRate /= difficulty;**.
4. Corrige el error en DifficultyButton.cs al pasar el parámetro de dificultad a **StartGame(difficulty)**.

```

1 referencia
public void StartGame(int difficulty)
{
    isGameActive = true;
    score = 0;
    spawnRate = spawnRate / difficulty;

    StartCoroutine(SpawnTarget());
    UpdateScore(0);

    titleScreen.gameObject.SetActive(false);
}

}

```