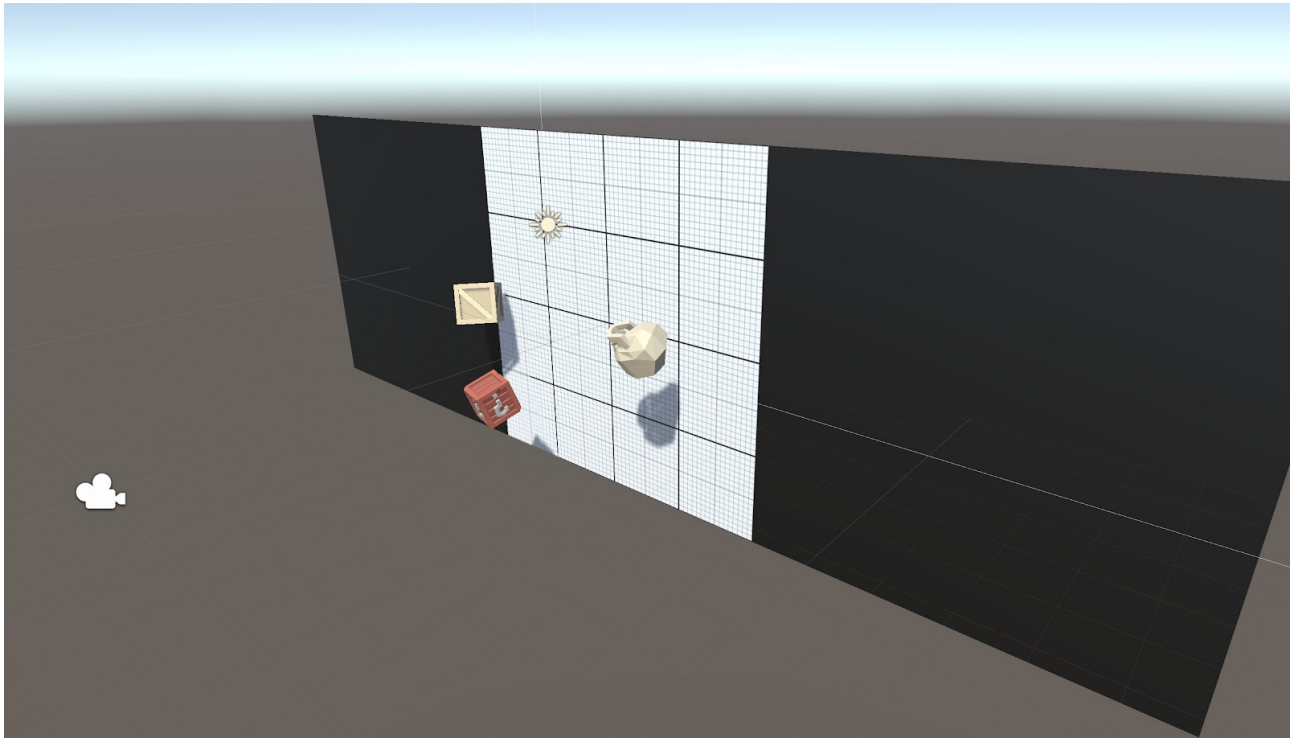
Prototipo 5: Clic de ratón

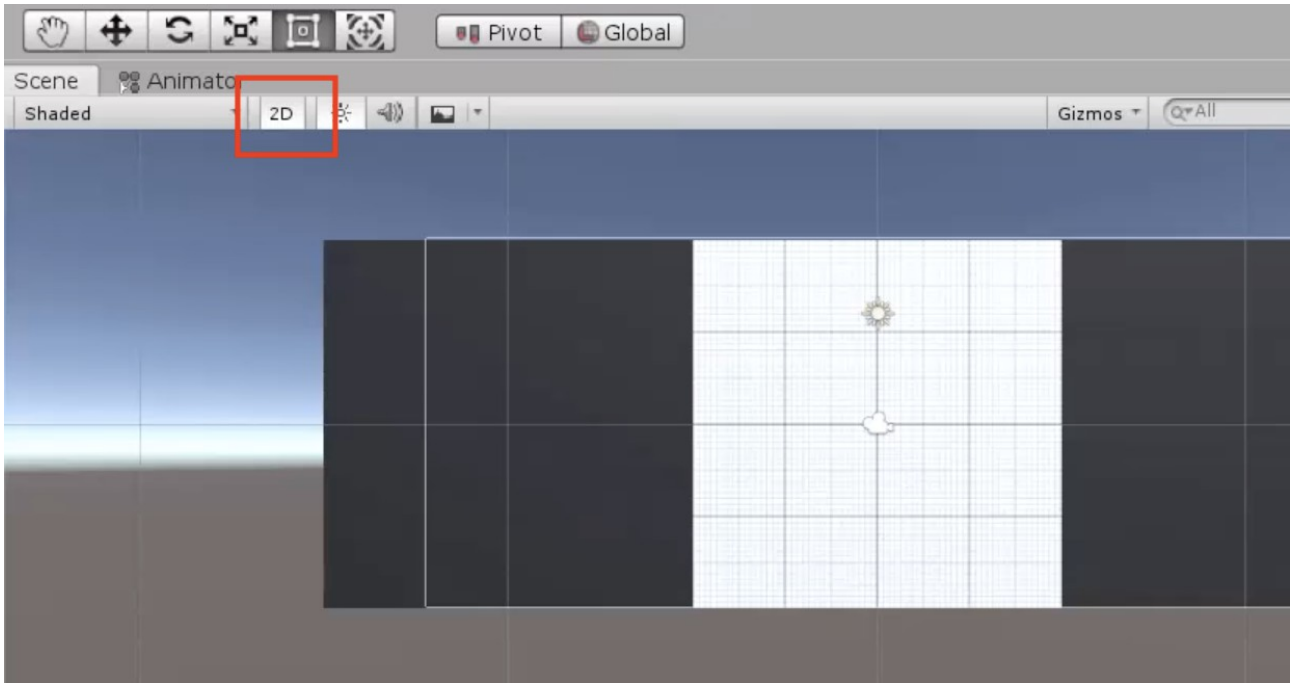


Comenzaremos creando un nuevo proyecto e importando los archivos iniciales, luego cambiaremos la vista del juego a 2D. A continuación haremos una lista de objetos **objetivos/target** en los que el jugador puede hacer clic: tres objetos "buenos" y uno "malo". Los objetivos se lanzarán girando en el aire después de aparecer en una posición aleatoria desde la parte inferior de la pantalla. El fin es permitir que el jugador los destruya con un clic.

1. Crea un nuevo proyecto y cambia a vista 2D

Necesitamos crear un nuevo proyecto y descargar los archivos iniciales para que todo esté en funcionamiento.

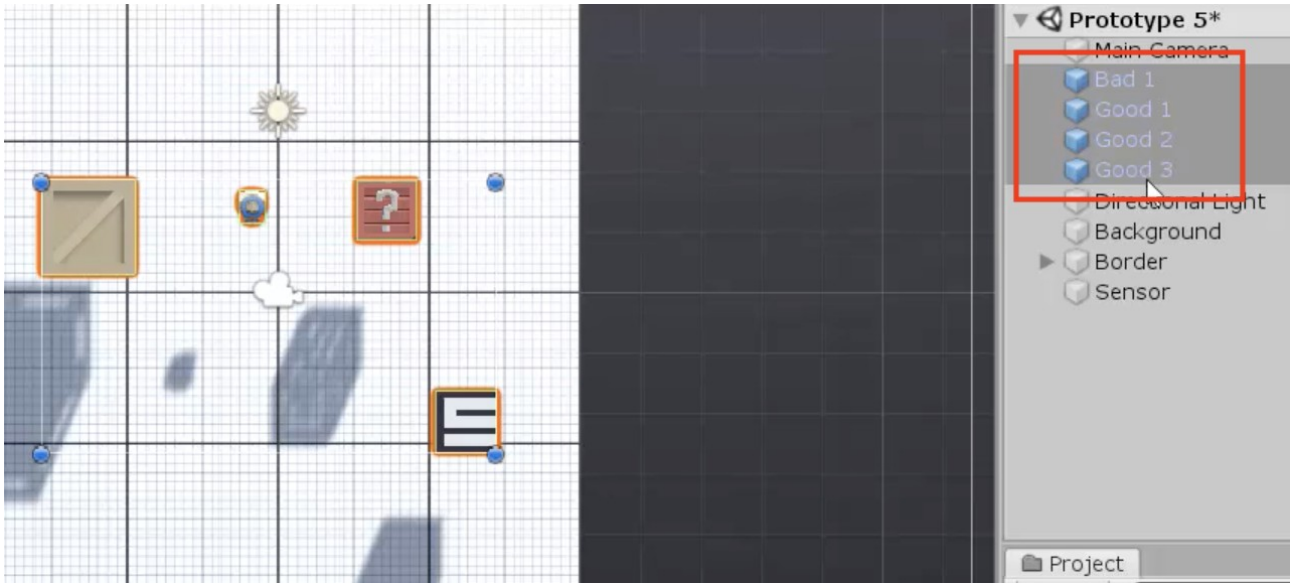
- Abre Unity Hub y crea un proyecto "***Prototipo 5***" vacío en el directorio del curso con la versión correcta de Unity.
- Descarga y extrae el contenido del fichero comprimido y luego importa el paquete .unity a tu proyecto.
- Abre la escena ***Prototype 5***, elimina la escena de muestra sin guardarla.
- Haz clic en el icono 2D en la vista de escena para poner la vista de escena en 2D.
- Opcionalmente puedes cambiar la textura y el color del fondo y el color de los bordes.



2. Crea objetivos buenos y malos

Lo primero que necesitamos en nuestro juego son tres objetos buenos para recoger y un objeto malo para evitar. Dependerá de ti decidir cuál es bueno y cuál es malo.

- Desde la biblioteca ***Library***, arrastra 3 objetos "buenos" y 1 objeto "malo" a la escena, cámbiales el nombre a ***"Good 1"***, ***"Good 2"***, ***"Good 3"***, y ***"Bad 1"***.
- Añade componentes ***Rigid Body*** y ***Box Collider*** a los objetos y asegúrate de que los Colliders rodeen los objetos correctamente.
- Crea una nueva carpeta ***Scripts*** y un nuevo script "***Target***" dentro de ella, añádelo a los objetos objetivo.
- Arrastra los 4 objetivos a la carpeta Prefabs para crear "***original prefabs***" y luego elimínalos de la escena.



3. Lanza objetos al aire al azar.
Ahora que tenemos 4 prefabs objetivo con el mismo script, debemos lanzarlos al aire con una fuerza, torsión y posición aleatorias.

- En ***Target.cs***, declara un nuevo ***private Rigidbody targetRb;*** e inicialízalo en Start()
- En Start(), agrega una fuerza hacia arriba multiplicada por una velocidad aleatoria
- Añade una torsión con valores xyz aleatorios
- Establece la posición con un valor X aleatorio

```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    // Start is called before the first frame update
    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);
    }

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

4. Reemplazamos el código desordenado con nuevos métodos.
En lugar de dejar que la fuerza, la rotación y la posición aleatorias hagan que nuestra función Start() sea confusa e ilegible, vamos a almacenar cada una de ellas en métodos personalizados nuevos y con nombres claros.

- Declaramos e inicializamos nuevas variables float privadas para ***minSpeed, maxSpeed, maxTorque, xRange, and ySpawnPos;***
- Crea una nueva función para ***Vector3 RandomForce()*** y llámala en el método Start().
- Crea una nueva función para ***float RandomTorque()*** y llámala en el método Start().
- Crea una nueva función para ***RandomSpawnPos()***, haz que devuelva un nuevo Vector3 y llámala en el método Start()

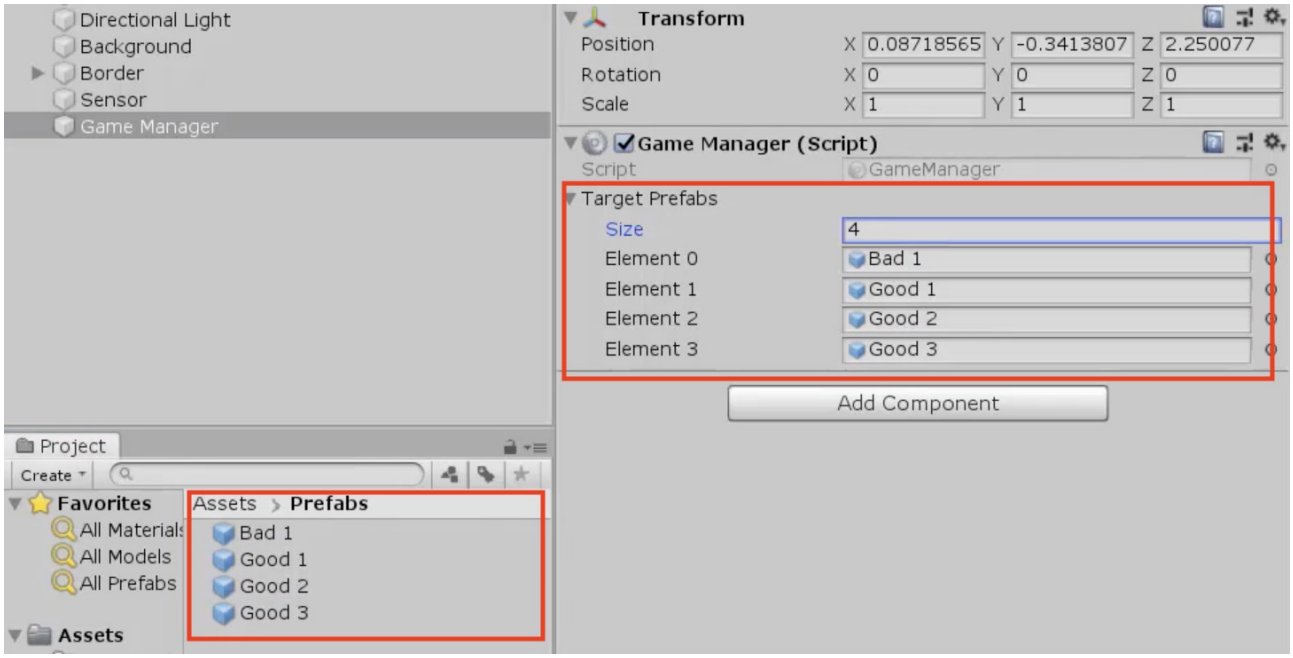
```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    private float minSpeed = 12;
    private float maxSpeed = 16;
    private float maxTorque = 10;
    private float xRange = 4;
    private float ySpawnPos = -6;
    // Start is called before the first frame update
    void Start()
    {
        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);
        targetRb.AddForce(RandomForce(), ForceMode.Impulse);
        targetRb.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);
    }
    float RandomTorque()
    {
        return Random.Range(-maxTorque, maxTorque);
    }
    Vector3 RandomSpawnPos()
    {
        return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
    }
}
```

5. Creamos una lista de objetos en Game Manager.

Lo siguiente que debemos hacer es crear una lista para que estos objetos se generen. En lugar de crear un ***Spawn Manager*** como en los prototipos anteriores, para estas funciones de generación crearemos un ***Game Manager*** que también controlará los estados del juego más adelante.

- Crea un nuevo objeto vacío "***Game Manager***" y añádele un nuevo script ***GameManager*** y luego ábrelo.
- Declara una nueva lista pública de objetivos ***public List<GameObject> targets;*** luego, en el inspector de Game Manager, cambia el tamaño de la lista a 4 y asigna los prefabs que habíamos escogido.



6. Crea una rutina para generar objetos.

Ahora que tenemos una lista de objetos prefabricados, debemos crear instancias de ellos en el juego usando corrutinas y un nuevo tipo de bucle.

- Declara e inicializa una nueva variable ***private float spawnRate***
- Crea un nuevo método ***IEnumerator SpawnTarget ()***
- Dentro del nuevo método, ***while(true),*** esperamos 1 segundo, generamos un índice aleatorio y creamos un objetivo de la lista con ese índice.
- En Start(), usa 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()
    {

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

7. Destruye cada objetivo con un clic o con el sensor.

Ahora que nuestros objetivos están apareciendo y siendo lanzados al aire, necesitamos una forma para que el jugador los destruya con un clic. También necesitamos destruir cualquier objetivo que caiga por debajo de la pantalla.

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

```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    private float minSpeed = 12;
    private float maxSpeed = 16;
    private float maxTorque = 10;
    private float xRange = 4;
    private float ySpawnPos = -6;
    // Start is called before the first frame update
    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);
        targetRb.AddForce(RandomForce(), ForceMode.Impulse);
        targetRb.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);
```

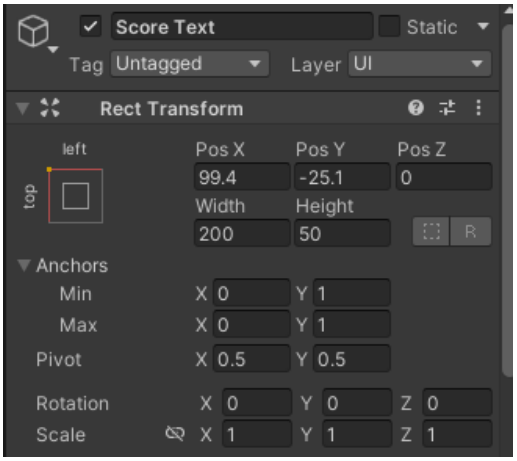
```
    }
    float RandomTorque()
    {
        return Random.Range(-maxTorque, maxTorque);
    }
    Vector3 RandomSpawnPos()
    {
        return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
    }
    private void OnMouseDown()
    {
        Destroy(gameObject);
    }
    private void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
    }
}
```

Manteniendo el marcador

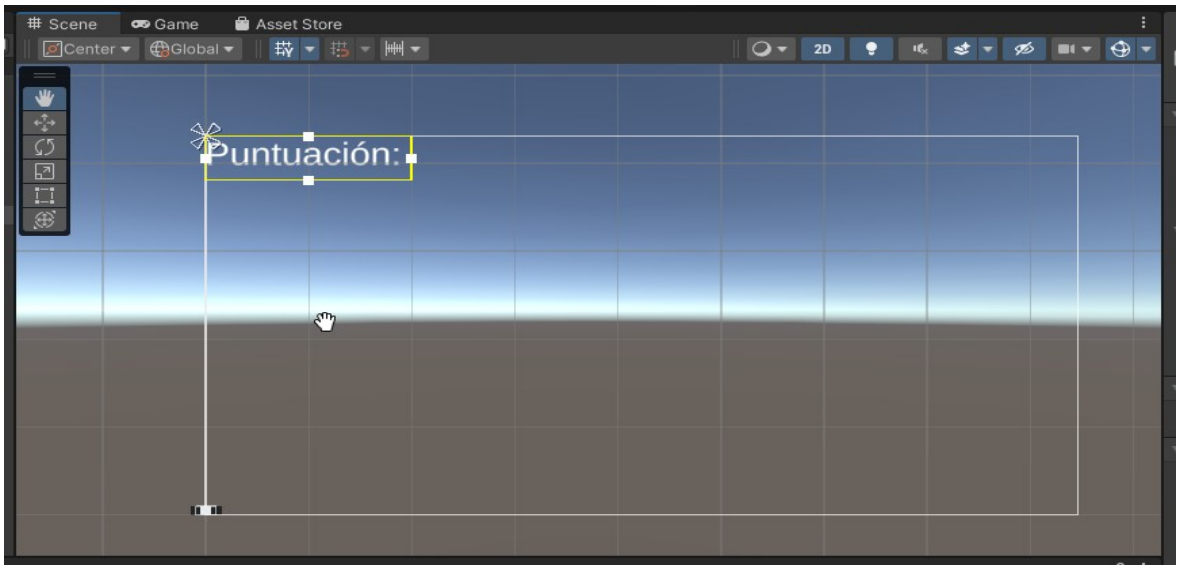
Los objetos vuelan en la escena y el jugador puede hacer clic para destruirlos, pero no pasa nada. Ahora vamos a mostrar una puntuación en la interfaz de usuario que mantenga y muestre los puntos del jugador. Le daremos a cada objeto un valor de puntos diferente, sumando o restando puntos al hacer clic. Por último, agregaremos explosiones cuando cada objetivo sea destruido.
Para llevarlo a cabo se mostrará una sección "Puntuación:" en la interfaz de usuario, comenzando en cero. Cuando el jugador hace clic en un objetivo, la puntuación se actualizará y las partículas explotarán a medida que se destruya el objetivo. Cada objetivo "Bueno" añade un valor de puntos diferente a la puntuación, mientras que el objetivo "Malo" resta de la puntuación.

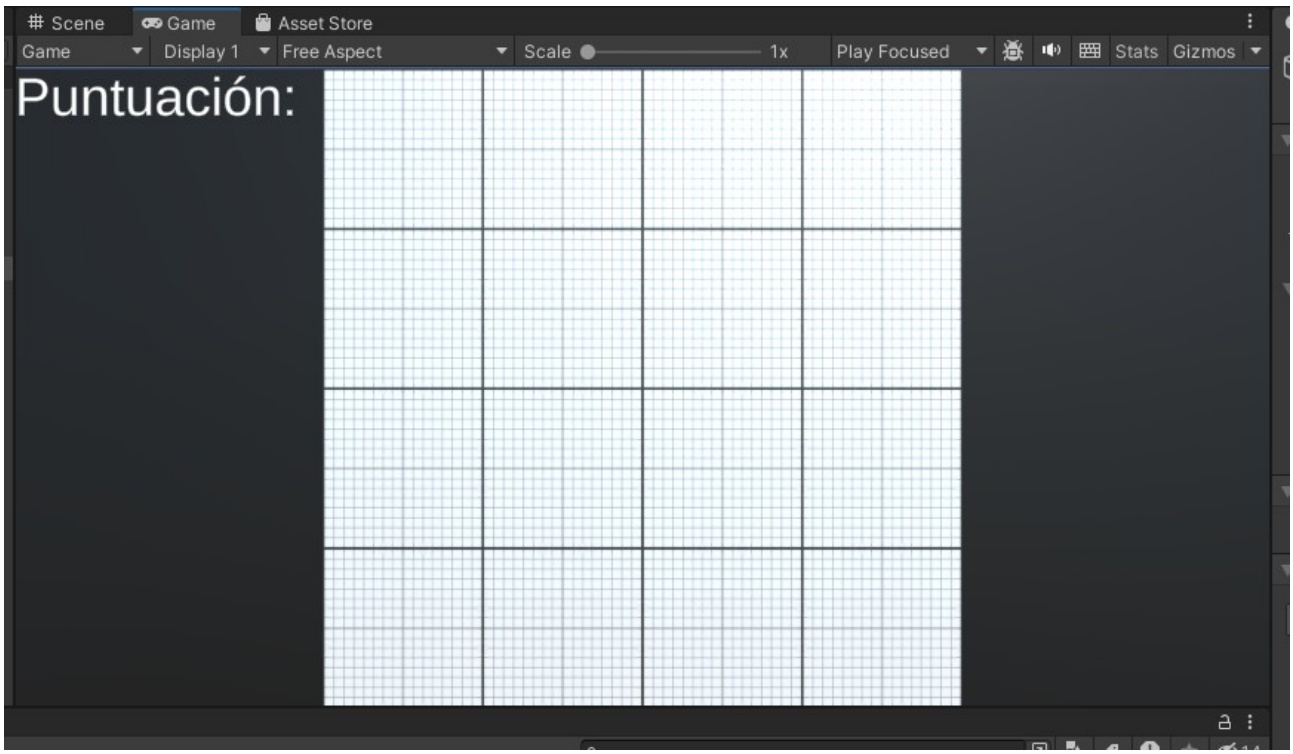
8. Agrega el marcador y colocalo en la pantalla.
Para mostrar la puntuación en pantalla, debemos incluir nuestro primer elemento de interfaz de usuario.

- En la ventana de jerarquía, haz ***Create > UI > TextMeshPro***, luego, si te lo solicita, haz clic en el botón para ***Import TMP Essentials***
- Cambia el nombre del nuevo objeto a "Score Text" y luego aléjalo (zoom out) para ver el canvas en la vista de escena.
- Cambia el punto de anclaje ***Anchor Point*** para que quede anclado a la esquina superior izquierda.



- En el inspector, cambie su Pos X y Pos Y para que esté en la esquina superior izquierda.

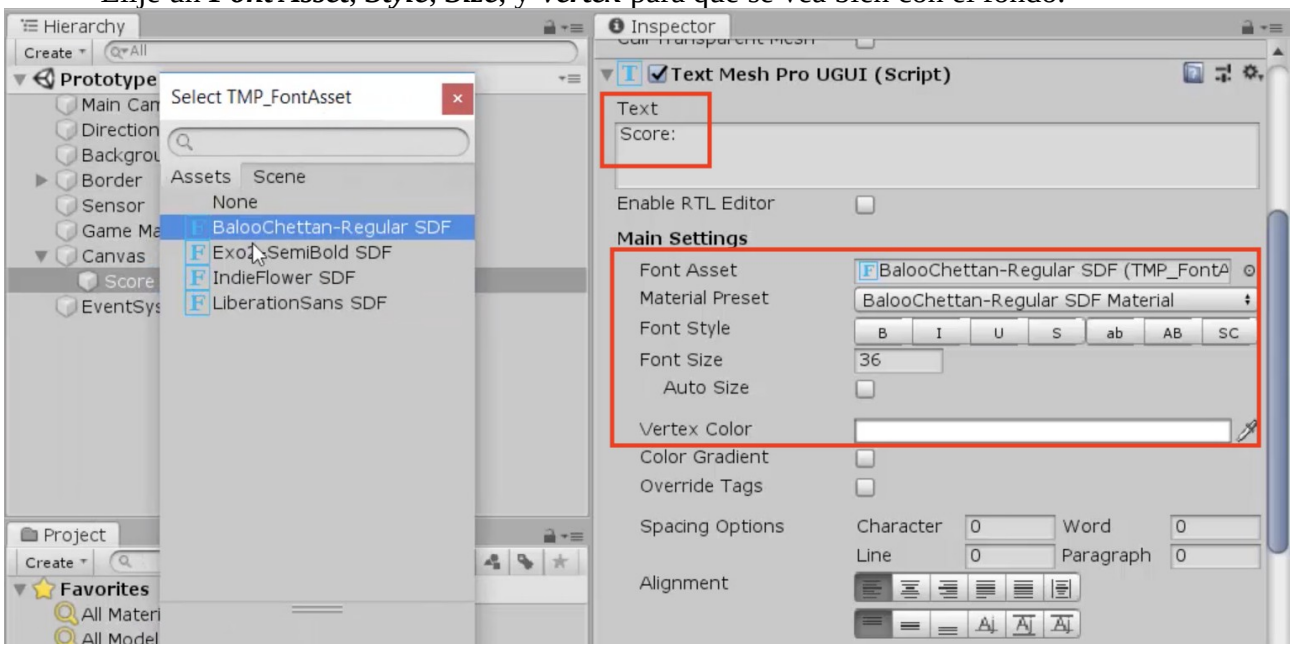




9. Edita las propiedades del texto del marcador.

Ahora que el texto básico está en la escena y posicionado correctamente, debemos editar sus propiedades para que se vea bien y tenga el texto correcto.

- Cambia el texto a "Score:"
- Elije un ***Font Asset***, ***Style***, ***Size***, y ***Vertex*** para que se vea bien con el fondo.



10. Inicializa el texto del marcador y la variable del mismo.

Tenemos un excelente lugar para mostrar la puntuación en la interfaz de usuario pero todavía no muestra nada. Necesitamos que la interfaz de usuario muestre una variable de puntuación, para que el jugador pueda realizar un seguimiento de sus puntos.

- En la parte superior de GameManager.cs, agrega "***using TMPro;***";
- Declara una nueva variable ***public TextMeshProUGUI scoreText*** y luego asígnale a esa variable en el inspector el ***Score Text***.

- Crea una nueva variable ***private int score*** e inicialízala en Start() como ***score = 0;***
- También en Start(), establece ***scoreText.text = "Puntuación: " + puntuación;***

```
public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(SpawnTarget());
        score = 0;
        scoreText.text = "Puntuación: " + score;
    }
```

11. Crea un nuevo método UpdateScore para mejorar el código.

El texto de la puntuación muestra la variable de puntuación perfectamente, pero nunca se actualiza. Necesitamos escribir una nueva función que acumule puntos para mostrar en la interfaz de usuario.

- Crea un nuevo método ***UpdateScore*** privado y vacío que requiera un parámetro ***int scoreToAdd***
- Corta y pega ***scoreText.text = "Score: " + score;*** en el nuevo método, luego llama al método UpdateScore(0) en Start()
- En ***UpdateScore()***, incremente la puntuación sumando ***score += scoreToAdd;***
- Llama al método ***UpdateScore(5)*** en la función ***spawnTarget()***

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

public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    // Start is called before the first frame update
    void Start()
    {
        StartCoroutine(SpawnTarget());
        score = 0;
        //scoreText.text = "Puntuación: " + score;
        UpdateScore(0);
    }

    // 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]);
        }
    }
    private void UpdateScore(int scoreToAdd)
    {
```

```
        score += scoreToAdd;
        scoreText.text = "Score: " + score;
    }
}
```

12. Añade puntuación cuando los objetivos sean destruidos.

Ahora que tenemos un método para actualizar la puntuación, debemos llamarlo en el script del Target cada vez que se destruya un objetivo.

- En GameManager.cs, haz público el método UpdateScore.
- En Target.cs, crea una referencia al GameManager ***private GameManager gameManager;***
- Inicializa GameManager en Start() usando el método Find()
- Cuando se destruye un objetivo, llama a ***UpdateScore(5);*** luego elimina la llamada al método de ***SpawnTarget()***

```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    private float minSpeed = 12;
    private float maxSpeed = 16;
    private float maxTorque = 10;
    private float xRange = 4;
    private float ySpawnPos = -6;
    private GameManager gameManager;
    // Start is called before the first frame update
    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);
        targetRb.AddForce(RandomForce(), ForceMode.Impulse);
        targetRb.AddTorque(RandomTorque(), RandomTorque(), RandomTorque(),
ForceMode.Impulse);
        transform.position = RandomSpawnPos();
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
    }
    // Update is called once per frame
    void Update()
    {
    }
    Vector3 RandomForce()
    {
        return Vector3.up * Random.Range(minSpeed, maxSpeed);
    }
    float RandomTorque()
    {
        return Random.Range(-maxTorque, maxTorque);
    }
    Vector3 RandomSpawnPos()
    {
        return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
    }
    private void OnMouseDown()
    {
        Destroy(gameObject);
        gameManager.UpdateScore(5);
    }
    private void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
    }
}
```

13. Añade puntuación cuando los objetivos sean destruidos.

La puntuación se actualiza cuando se hace clic en los objetivos, pero queremos darle a cada uno de los objetivos un valor diferente. Los objetos buenos deben variar en valor de puntos y el objeto malo debe restar puntos.

- En Target.cs, crea una nueva variable pública ***public int pointValue***
- En cada uno de los inspectores de los prefabs de los Target, establece el valor en puntos que valga cada uno, incluido el valor negativo del objetivo malo.
- Agrega la nueva variable a ***UpdateScore(pointValue);***

```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    private float minSpeed = 12;
    private float maxSpeed = 16;
    private float maxTorque = 10;
    private float xRange = 4;
    private float ySpawnPos = -6;
    private GameManager gameManager;
    public int pointValue;
    // Start is called before the first frame update
    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);
        targetRb.AddForce(RandomForce(), ForceMode.Impulse);
        targetRb.AddTorque(RandomTorque(), RandomTorque(), RandomTorque(),
ForceMode.Impulse);
        transform.position = RandomSpawnPos();
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
    }
    // Update is called once per frame
    void Update()
    {
    }
    Vector3 RandomForce()
    {
        return Vector3.up * Random.Range(minSpeed, maxSpeed);
    }
    float RandomTorque()
    {
        return Random.Range(-maxTorque, maxTorque);
    }
    Vector3 RandomSpawnPos()
    {
        return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
    }
    private void OnMouseDown()
    {
        Destroy(gameObject);
        //gameManager.UpdateScore(5);
        gameManager.UpdateScore(pointValue);
    }
    private void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
    }
}
```

14. Agrega una explosión de partículas.

La puntuación es totalmente funcional, pero hacer clic en los objetivos es algo insatisfactorio. Para darle vida a las cosas, vamos a agregar algunas partículas explosivas cada vez que se haga clic en un objetivo:

- En Target.cs, agrega una nueva variable pública ***public ParticleSystem explosionParticle***
- Para cada uno de los prefabs objetivo, asigna un prefab ***particle prefab*** desde ***Course Library*** > ***Particles*** a la variable ***Explosion Particle*** que acabamos de crear.
- En la función ***OnMouseDown()*** debemos instanciar los prefab de exploxión.

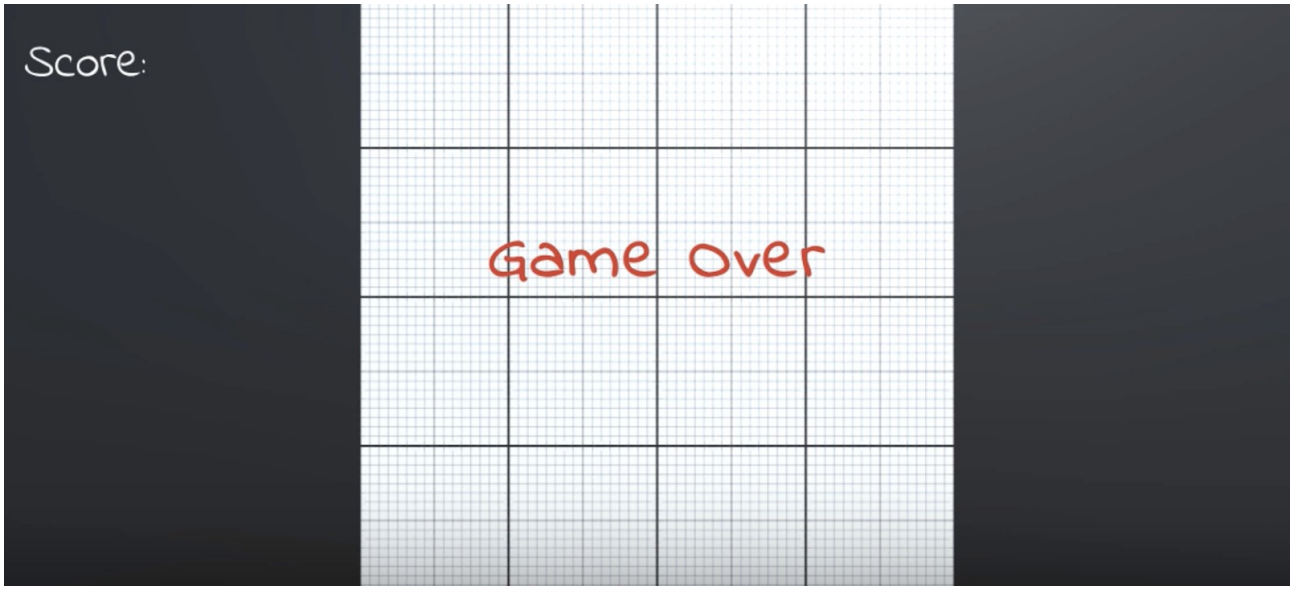
```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    private float minSpeed = 12;
    private float maxSpeed = 16;
    private float maxTorque = 10;
    private float xRange = 4;
    private float ySpawnPos = -6;
    private GameManager gameManager;
    public int pointValue;
    public ParticleSystem explosionParticle;
    // Start is called before the first frame update
    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);
        targetRb.AddForce(RandomForce(), ForceMode.Impulse);
        targetRb.AddTorque(RandomTorque(), RandomTorque(), RandomTorque(),
ForceMode.Impulse);
        transform.position = RandomSpawnPos();
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
    }
    // Update is called once per frame
    void Update()
    {
    }
    Vector3 RandomForce()
    {
        return Vector3.up * Random.Range(minSpeed, maxSpeed);
    }
    float RandomTorque()
    {
        return Random.Range(-maxTorque, maxTorque);
    }
    Vector3 RandomSpawnPos()
    {
        return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
    }
    private void OnMouseDown()
    {
        Destroy(gameObject);
        Instantiate(explosionParticle,
transform.position,explosionParticle.transform.rotation);
        //gameManager.UpdateScore(5);
        gameManager.UpdateScore(pointValue);
    }
    private void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
    }
}
```

Game Over

Hemos añadido un contador de puntuación al juego, pero hay muchos otros elementos de interfaz de usuario que podríamos agregar. Ahora vamos a crear un texto de "Game Over" que se mostrará cuando un objetivo "bueno" caiga por debajo del sensor. Cuando finalice el juego, los objetivos dejarán de aparecer y la puntuación se restablecerá. Al final, añadiremos un botón "Restart Game" que permitirá al jugador reiniciar el juego después de haber perdido.
Para realizar esta tarea, cuando un objetivo "bueno" caiga debajo del sensor en la parte inferior de la pantalla, los objetivos dejarán de aparecer y aparecerá un mensaje de "Game Over". Justo debajo del mensaje "Game Over" habrá un botón "Restart Game" que reiniciará el juego y restablecerá la puntuación, para que el jugador pueda disfrutarlo todo nuevamente.

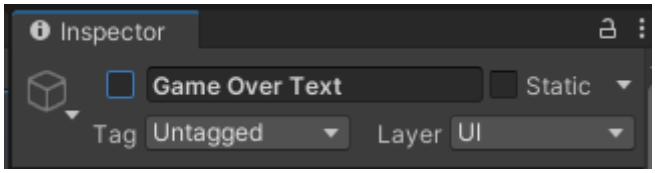
15. Crea un objeto de texto Game Over.
Si queremos que aparezca un texto de "Game Over" cuando finalice el juego, lo primero que haremos será crear y personalizar un nuevo elemento de texto de la interfaz de usuario que diga "Game Over".

- Haz clic derecho en el Canvas y crea un ***UI > TextMeshPro - Text*** y cámbiale el nombre a "***Game Over Text***".
- En el inspector, edita las propiedades ***Text, Pos X, Pos Y, Font Asset, Size, Style, Color***, y ***Alignment***.
- Establece la configuración de "***Wrapping***" a "***Disabled***".



16. Haz que aparezca el texto de GameOver.
Tenemos el texto de Game Over en la pantalla, pero en este momento simplemente está fijo y bloqueando nuestra vista. Deberíamos desactivarlo, para que pueda volver a aparecer cuando finalice el juego.

- En GameManager.cs, crea una nueva variable ***public TextMeshProUGUI gameOverText;*** y asignale el objeto ***Game Over*** en el inspector
- Desactiva el texto Game Over de forma predeterminada.



- En Start(), activa el texto Game Over para ver si funciona

```
public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI gameOverText;
    // Start is called before the first frame update
    void Start()
    {
        gameOverText.gameObject.SetActive(true);
...
```

17. Crea la función GameOver.

Hemos hecho que aparezca temporalmente el texto "Game Over" al comienzo del juego, pero en realidad queremos activarlo cuando uno de los objetos "buenos" se pierda y caiga.

- Crea una nueva función ***public void GameOver()*** y mueve el código que activa el texto Game Over dentro de ella.
- En Target.cs, llama a ***gameManager.GameOver()*** si un objetivo choca con el sensor.
- Agrega una nueva etiqueta "***Bad***" al objeto malo y añade una condición de forma que solo se activará el Game Over del juego si no es un objeto malo

```
public class Target : MonoBehaviour
{
    private Rigidbody targetRb;
    private float minSpeed = 12;
    private float maxSpeed = 16;
    private float maxTorque = 10;
    private float xRange = 4;
    private float ySpawnPos = -6;
    private GameManager gameManager;
    public int pointValue;
    public ParticleSystem explosionParticle;
    // Start is called before the first frame update
    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);
        targetRb.AddForce(RandomForce(), ForceMode.Impulse);
        targetRb.AddTorque(RandomTorque(), RandomTorque(), RandomTorque(),
ForceMode.Impulse);
        transform.position = RandomSpawnPos();
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
    }
    // Update is called once per frame
    void Update()
    {
    }
    Vector3 RandomForce()
    {
        return Vector3.up * Random.Range(minSpeed, maxSpeed);
    }
    float RandomTorque()
    {
        return Random.Range(-maxTorque, maxTorque);
    }
    Vector3 RandomSpawnPos()
```

```
    {
        return new Vector3(Random.Range(-xRange, xRange), ySpawnPos);
    }
    private void OnMouseDown()
    {
        Destroy(gameObject);
        Instantiate(explosionParticle, transform.position,
explosionParticle.transform.rotation);
        //gameManager.UpdateScore(5);
        gameManager.UpdateScore(pointValue);
    }
    private void OnTriggerEnter(Collider other)
    {
        Destroy(gameObject);
        if (!gameObject.CompareTag("Bad"))
        {
            gameManager.GameOver();
        }
    }
}
```

18. Deja de generar objetivos y detén el marcador cuando se llega a GameOver

El mensaje "Game Over" aparece exactamente cuando queremos que aparezca, pero el juego continúa. Para detener realmente el juego, debemos dejar de generar objetivos y dejar de actualizar la puntuación del jugador.

- Crea una nueva variable booleana
- Como primera línea en Start(), establece ***isGameActive = true;*** y en GameOver(), cambia el valor a ***isGameActive = false;***
- Para evitar seguir generando objetivos, en la rutina ***SpawnTarget(),*** cambia ***while (true)*** a ***while (isGameActive)***.
- Para evitar que la puntuación siga actualizándose, en Target.cs, en la función ***OnMouseDown(),*** añade la condición ***if (gameManager.isGameActive) {***

Archivo GameManager

```
public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI gameOverText;
    public bool isGameActive;
    // Start is called before the first frame update
    void Start()
    {
        isGameActive = true;
        StartCoroutine(SpawnTarget());
        score = 0;
        //scoreText.text = "Puntuación: " + score;
        UpdateScore(0);
        //gameOverText.gameObject.SetActive(true);
    }

    // Update is called once per frame
    void Update()
    {
    }
    IEnumerator SpawnTarget()
    {
        //while (true)
        while (isGameActive)
        {
```

```
            yield return new WaitForSeconds(spawnRate);
            int index = Random.Range(0, targets.Count);
            Instantiate(targets[index]);
        }
    }
    //private void UpdateScore(int scoreToAdd)
    public void UpdateScore(int scoreToAdd)
    {
        score += scoreToAdd;
        scoreText.text = "Score: " + score;
    }
    public void GameOver()
    {
        isGameActive = false;
        gameOverText.gameObject.SetActive(true);
    }
}
```

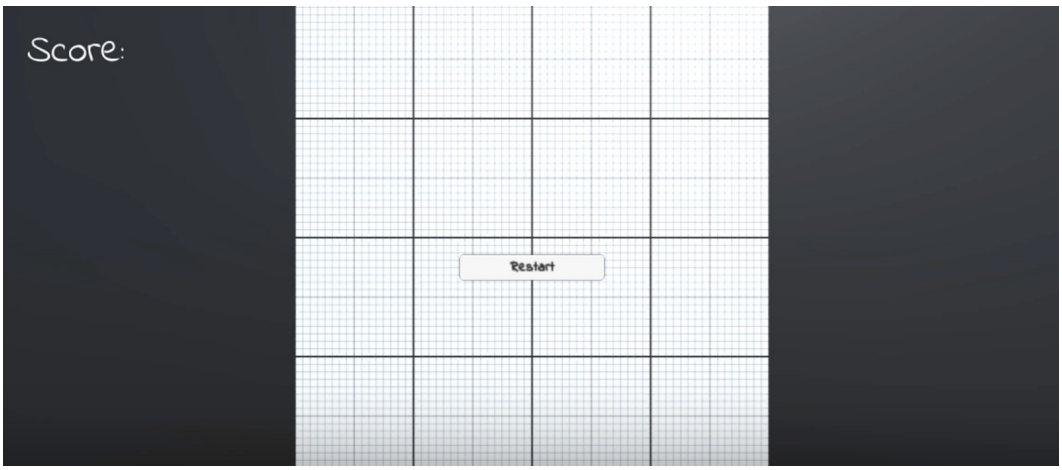
Archivo Target

```
 private void OnMouseDown()
 {
     if (gameManager.isGameActive)
     {
         Destroy(gameObject);
         Instantiate(explosionParticle, transform.position,
explosionParticle.transform.rotation);
         //gameManager.UpdateScore(5);
         gameManager.UpdateScore(pointValue);
     }
 }
```

19. Mostramos un botón de reinicio

Nuestras mecánicas de Game Over funcionan, pero no hay forma de volver a jugar el juego. Para permitir que el jugador reinicie el juego, crearemos nuestro primer botón de interfaz de usuario.

- Haz clic derecho en el Canvas y a continuación ***Create*** > ***UI*** > ***Button***
  Nota: También puedes usar ***Button - TextMeshPro*** para tener más control sobre el texto del botón.
- Cambia el nombre del botón a "Restart Button"
- Reactiva temporalmente el texto Game Over para reposicionar el botón de reinicio correctamente con el texto, luego desactívalo nuevamente.
- Selecciona el objeto secundario Text y edita su texto para que diga "Restart", su tipo de fuente, estilo y tamaño.



20. Haz que el botón de reinicio funcione

Hemos añadido el botón de reinicio a la escena y parece bien, pero ahora necesitamos hacerlo funcionar y reiniciar el juego.

- En GameManager.cs, añade ***using UnityEngine.SceneManagement;***
- Crea una nueva función pública vacía ***public void RestartGame()*** para que se recargue la escena actual
- En el inspector del botón, haga clic en + para agregar un nuevo evento ***On Click event***, arrástralo al objeto Game Manager y selecciona la función ***GameManager.RestartGame***.

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

public class GameManager : MonoBehaviour
{
....
    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

21. Muestra el botón de reinicio al terminar el juego

El botón de reinicio se ve, pero no lo queremos en pantalla durante todo el juego.

De manera similar al mensaje "Game Over", desactivaremos el botón Restart mientras el juego esté activo.

- En la parte superior de GameManager.cs añade ***using UnityEngine.UI;***
- Declara un nuevo botón público ***public Button restartButton;*** y asígnale el botón ***Restart Button*** en el inspector
- Desmarca la casilla de verificación “Active” para el botón Restart Button en el inspector.
- En la función GameOver, activa el ***Restart Button***

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

public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI gameOverText;
    public bool isGameActive;
    public Button restartButton;
    // Start is called before the first frame update
    void Start()
    {
        isGameActive = true;
        StartCoroutine(SpawnTarget());
        score = 0;
        //scoreText.text = "Puntuación: " + score;
        UpdateScore(0);
        //gameOverText.gameObject.SetActive(true);
```

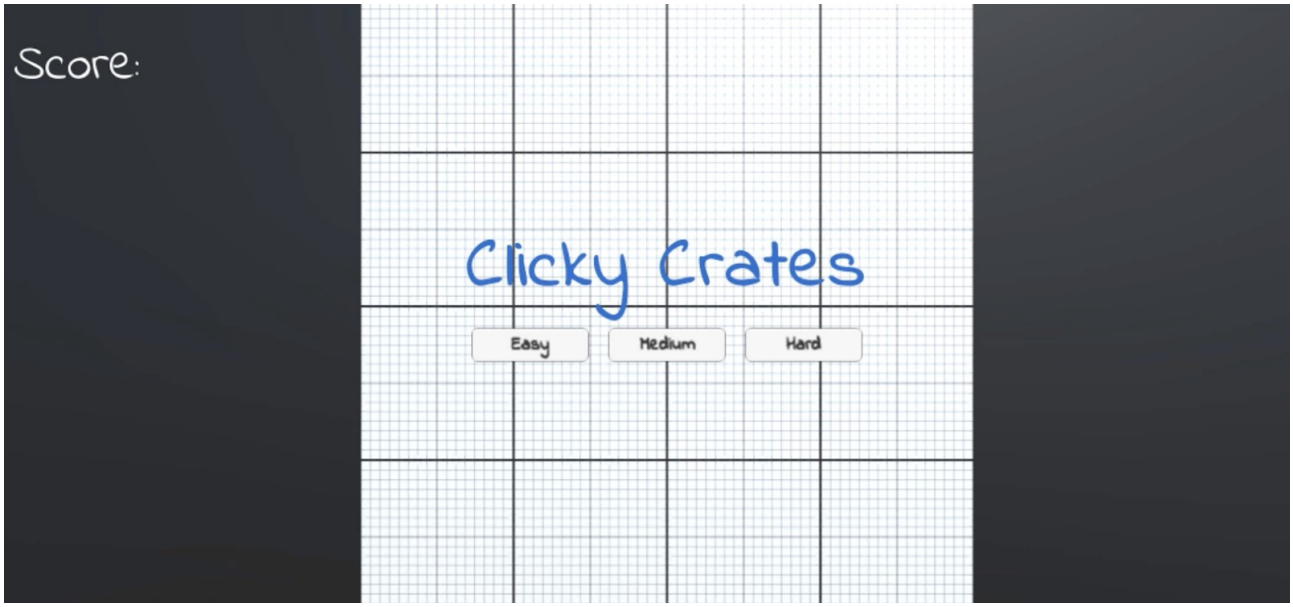
```
    }

    // Update is called once per frame
    void Update()
    {
    }
    IEnumerator SpawnTarget()
    {
        //while (true)
        while (isGameActive)
        {
            yield return new WaitForSeconds(spawnRate);
            int index = Random.Range(0, targets.Count);
            Instantiate(targets[index]);
        }
    }
    //private void UpdateScore(int scoreToAdd)
    public void UpdateScore(int scoreToAdd)
    {
        score += scoreToAdd;
        scoreText.text = "Score: " + score;
    }
    public void GameOver()
    {
        isGameActive = false;
        gameOverText.gameObject.SetActive(true);
        restartButton.gameObject.SetActive(true);
    }
    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

¿Cuál es la dificultad?

Para terminar nuestro juego, vamos a añadir una especie de menú y una pantalla de título. Vamos a crear nuestro propio título y le daremos estilo al texto para que se vea bien. Crearemos tres nuevos botones que establecerán la dificultad del juego. Cuanto mayor sea la dificultad, más rápido aparecerán los objetivos.
Para ello, al iniciar el juego, se abrirá un menú con el título en un lugar destacado y tres botones de dificultad en la parte inferior de la pantalla.
Cada dificultad afectará a la tasa de generación de los objetivos, aumentando la habilidad necesaria para evitar que caigan los objetivos "buenos".

22. Crea el texto del título y los botones de menú.
Lo primero que debemos hacer es crear todos los elementos de la interfaz de usuario que vamos a necesitar. Esto incluye un título grande, así como tres botones de dificultad.

- Duplica tu texto Game Over para crear tu texto de título ***Title***, editando su nombre, texto y todos sus atributos.
- Duplica el botón de reinicio y edita sus atributos para crear un botón para el nivel fácil "Easy Button"
- Edita y duplica el nuevo botón fácil para crear un “Medium Button” y un “Hard Button”.



23. Agregamos un script de DifficultyButton.
Nuestros botones de dificultad se ven enpantalla, pero en realidad no hacen nada. Si van a tener una funcionalidad personalizada, primero debemos darles un nuevo script.

- Para los 3 botones nuevos, en el componente ***Button***, en la sección ***On Click ()***, haz clic en el botón menos (-) para eliminar la funcionalidad ***RestartGame.***
- Crea un nuevo script ***DifficultyButton.cs*** y añádelo a los 3 botones.
- Añade al principio del fichero ***using UnityEngine.UI;*** en el apartado de las importaciones
- Crea una nueva variable ***private Button button;*** e inicializala en Start()

```
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()
    {

    }
}
```

24. Hacemos la llamada a SetDifficulty al hacer clic en el botón.

Ahora que tenemos un script para nuestros botones, podemos crear un método SetDifficulty y vincular ese método al clic de esos botones.

- Crea una nueva función vacía ***void SetDifficulty*** y, dentro de ella añade ***Debug.Log(gameObject.name + " was clicked");*** para comprobar el funcionamiento de los botones
- Añadimos el ***button listener*** para llamar a la función ***SetDifficulty***.

```
public class DifficultyButton : MonoBehaviour
{
    private Button button;
    // Start is called before the first frame update
    void Start()
    {
        button = GetComponent<Button>();
        button.onClick.AddListener(SetDifficulty);
    }

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

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

25. Ahora vamos a hacer que los botones inicien el juego.

Necesitamos una función StartGame que pueda comunicarse con SetDifficulty.

- En ***GameManager.cs***, crea una nueva función ***public void StartGame()*** y mueve todo desde ***Start()*** a ella.
- En ***DifficultyButton.cs***, crea una nueva variable ***private GameManager gameManager;*** e inicializala en ***Start()***
- En la función ***SetDifficulty()***, llama a ***gameManager.startGame();***

`GameManager.cs`

```
public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI gameOverText;
    public bool isGameActive;
    public Button restartButton;
    // Start is called before the first frame update
    void Start()
```

```
    {
        //isGameActive = true;
        //StartCoroutine(SpawnTarget());
        //score = 0;
        //scoreText.text = "Puntuación: " + score;
        //UpdateScore(0);
        //gameOverText.gameObject.SetActive(true);
    }

    // Update is called once per frame
    void Update()
    {
    }
    IEnumerator SpawnTarget()
    {
        //while (true)
        while (isGameActive)
        {
            yield return new WaitForSeconds(spawnRate);
            int index = Random.Range(0, targets.Count);
            Instantiate(targets[index]);
        }
    }
    //private void UpdateScore(int scoreToAdd)
    public void UpdateScore(int scoreToAdd)
    {
        score += scoreToAdd;
        scoreText.text = "Score: " + score;
    }
    public void GameOver()
    {
        isGameActive = false;
        gameOverText.gameObject.SetActive(true);
        restartButton.gameObject.SetActive(true);
    }
    public void RestartGame()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
    public void StartGame()
    {
        isGameActive = true;
        score = 0;
        StartCoroutine(SpawnTarget());
        UpdateScore(0);
    }
}
```

DifficultyButton.cs

```
public class DifficultyButton : MonoBehaviour
{
    private Button button;
    private GameManager gameManager;
    // Start is called before the first frame update
    void Start()
    {
        button = GetComponent<Button>();
        button.onClick.AddListener(SetDifficulty);
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
    }

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

```
    }
    void SetDifficulty()
    {
        Debug.Log(gameObject.name + " was clicked");
        gameManager.StartGame();
    }
}
```

26. Desactivamos la pantalla de título en StartGame.

Si queremos que la pantalla de título desaparezca al iniciar el juego, deberíamos guardar los elementos que la componen en un objeto vacío y desactivarlo en lugar de desactivar individualmente los elementos uno a uno.

- Haz clic derecho en el objeto ***Canvas*** hacemos ***Create*** > ***Empty Object***, le cambiamos el nombre a "***Title Screen***" y arrastramos los 3 botones y el título a él.
- En ***GameManager.cs***, crea una nueva variable ***public GameObject titleScreen;*** y asígnale el objeto en el inspector.
- En StartGame(), desactiva el objeto de pantalla de título.

```
public class GameManager : MonoBehaviour
{
    public List<GameObject> targets;
    private float spawnRate = 1.0f;
    private int score;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI gameOverText;
    public bool isGameActive;
    public Button restartButton;
    public GameObject titleScreen;
    // Start is called before the first frame update

…

    public void StartGame()
    {
        isGameActive = true;
        score = 0;
        StartCoroutine(SpawnTarget());
        UpdateScore(0);
        titleScreen.gameObject.SetActive(false);
    }
}
```

27. Vamos a utilizar un parámetro para cambiar la dificultad.

Los botones de dificultad inician el juego, pero aún así no cambian la dificultad del juego. Lo último que tenemos que hacer es conseguir que los botones de dificultad afecten la velocidad con la que aparecen los objetos objetivo.

- En ***DifficultyButton.cs***, crea una nueva variable de dificultad ***public int difficulty.*** Una vez hecho esto, en el Inspector, asigna la dificultad del botón fácil a 1, del botón medio a 2 y del botón difícil a 3.
- En el script ***GameManager.cs***, añade un parámetro ***int difficulty*** en la función ***StartGame()***
- En ***StartGame()***, hacemos que ***spawnRate /= difficulty;***
- Corregimos el error en ***DifficultyButton.cs*** pasando el parámetro de dificultad a ***StartGame(difficulty)***

Fichero GameManager.cs

```
public void StartGame(int difficulty)
{
    spawnRate /= difficulty;
    isGameActive = true;
    score = 0;
    StartCoroutine(SpawnTarget());
    UpdateScore(0);
    titleScreen.gameObject.SetActive(false);
}
```

Fichero DifficultyButton.cs

```
public class DifficultyButton : MonoBehaviour
{
    private Button button;
    private GameManager gameManager;
    public int difficulty;
    // Start is called before the first frame update
    void Start()
    {
        button = GetComponent<Button>();
        button.onClick.AddListener(SetDifficulty);
        gameManager = GameObject.Find("Game Manager").GetComponent<GameManager>();
    }

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

    }
    void SetDifficulty()
    {
        Debug.Log(gameObject.name + " was clicked");
        gameManager.StartGame(difficulty);
    }
}
```