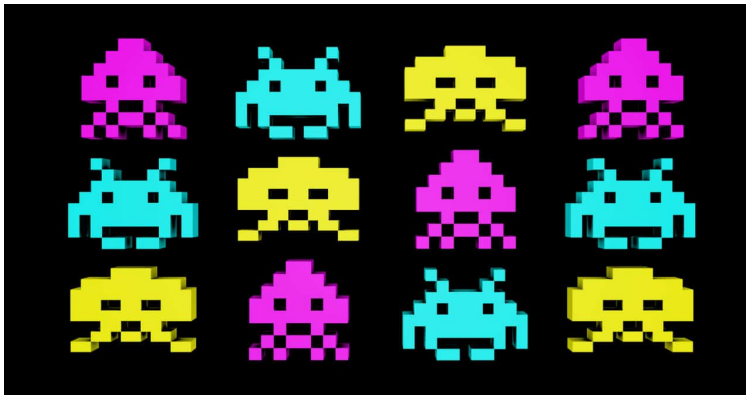
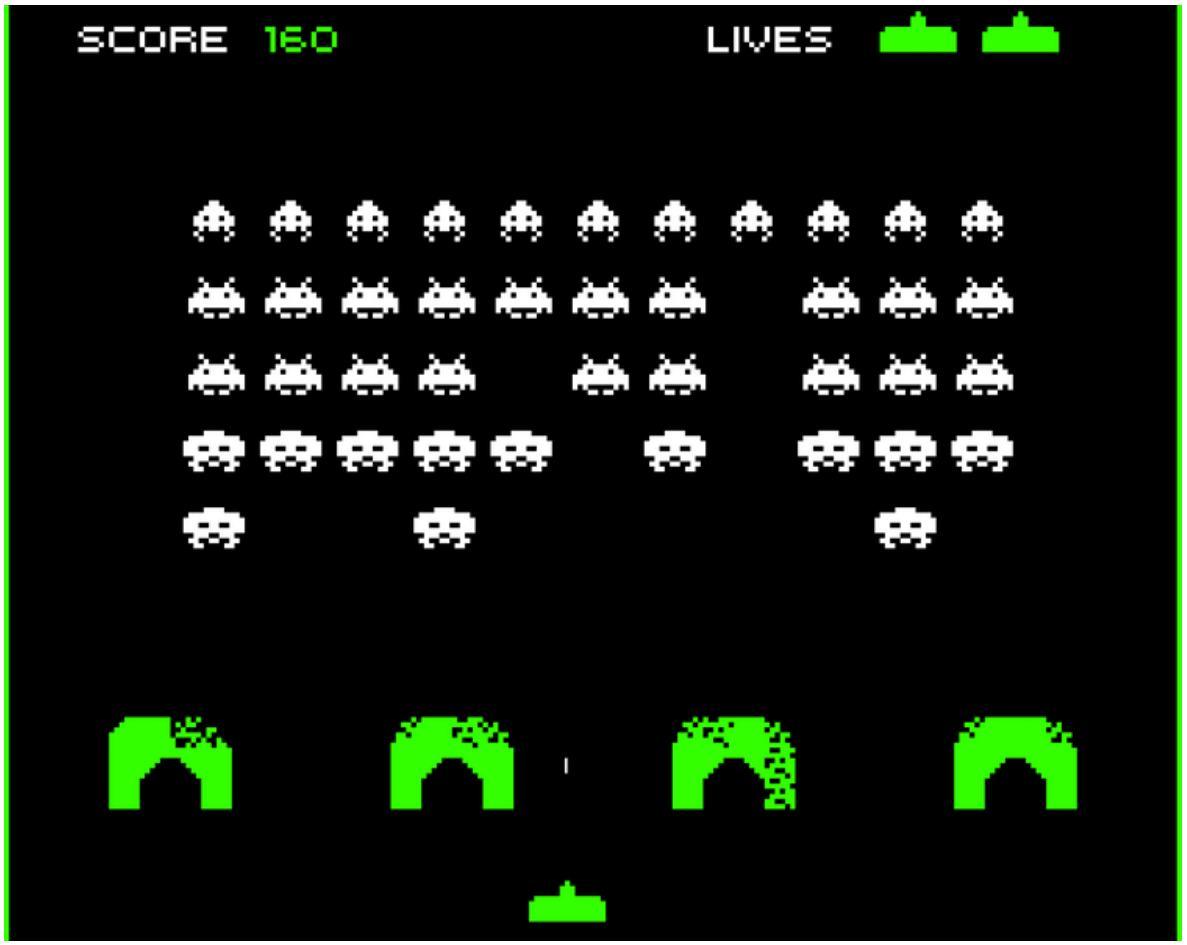
Clase 03 - Space Invaders



1- Qué es Space Invaders

Space Invaders, conocido en Japón como *Supēsuinbēdā*, es uno de los juegos retro más conocidos en el mundo. Lanzado a las salas de arcade en 1978 por la compañía japonesa de videojuegos Taito, rápidamente se convirtió en un gran éxito.



El cañón que dispara balas láser representa al jugador. El jugador puede esconderse detrás de cualquiera de las cuatro torchka, también conocidas como pillboxes.

Hay tres tipos de invasores: cangrejo, calamar y pulpo. Aparecen en un enjambre de múltiples filas y se mueven desde la parte superior hasta la inferior de la pantalla. El invasor cangrejo se ha convertido en un símbolo icónico universalmente asociado con las salas de arcade y los juegos en general.

El objetivo del juego es eliminar a los invasores antes de que lleguen a la parte inferior de la pantalla mientras se esquiva y se esconde detrás de las torchkas. Además del enjambre, también hay un OVNI que aparece de vez en cuando. Sin embargo, te centrarás en el enjambre.

En esta clase, replicaremos las características principales del juego de Space Invaders usando Unity. En el camino, aprenderás cómo:

- Generar y mover el enjambre de invasores.
- Hacer que los invasores en la cabeza del enjambre disparen balas láser.
- Moverte y disparar como jugador.
- Cambiar el tempo de la música y la velocidad del enjambre en base al conteo de bajas.

¡Hora de empezar!

2- Cómo empezar el proyecto

Nos vamos a descargar el proyecto desde el campus. El proyecto contiene algunas carpetas para ayudarte a comenzar. Abre Assets/RW y encontrarás los siguientes directorios:

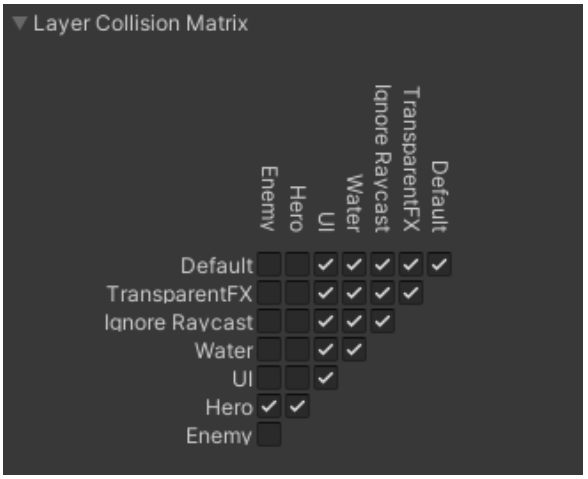
- **Animaciones:** Contiene animaciones prehechas y el animador para los prefabs de bala y explosión.
- **Prefabs:** Tiene todos los prefabs que usa el proyecto.
- **Recursos:** Contiene fuentes y texturas.
- **Escenas:** Contiene la escena principal con la que trabajarás.
- **Scripts:** Tiene todos los scripts del proyecto. La mayoría de ellos son estructuras vacías que llenarás.
- **Sonidos:** Tiene todos los efectos de sonido y el archivo de música.
- **Sprites:** Contiene todo el arte pixelado para el proyecto.

Navega a RW/Escenas y abre Principal. Haz clic en Play y verás el cañón del jugador rojo en la parte inferior central de la pantalla y escucharás el bucle de música. Además, verás las etiquetas de interfaz de usuario de Puntuación y Vidas.

Aún no puedes interactuar con el juego, pero añadirás la interactividad en este tutorial.

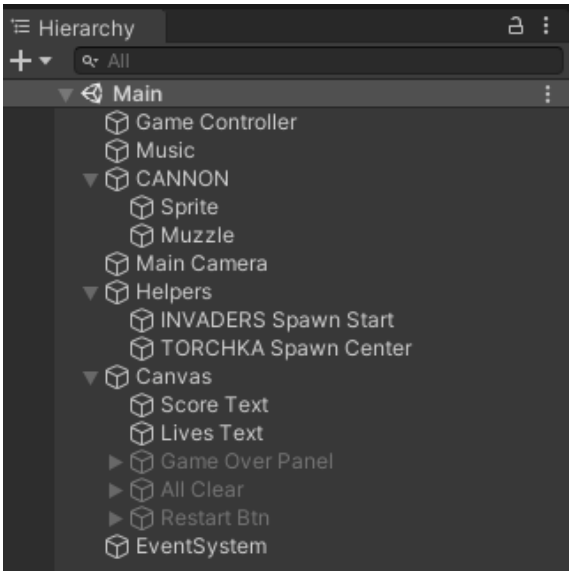
3- Configuración del Juego

Detén el juego y selecciona **Editar ▸ Configuración del Proyecto ▸ Físicas 2D**. Echa un vistazo a la Matriz de Colisión de Capas. Nota que este proyecto tiene dos capas personalizadas: Héroe y Enemigo.

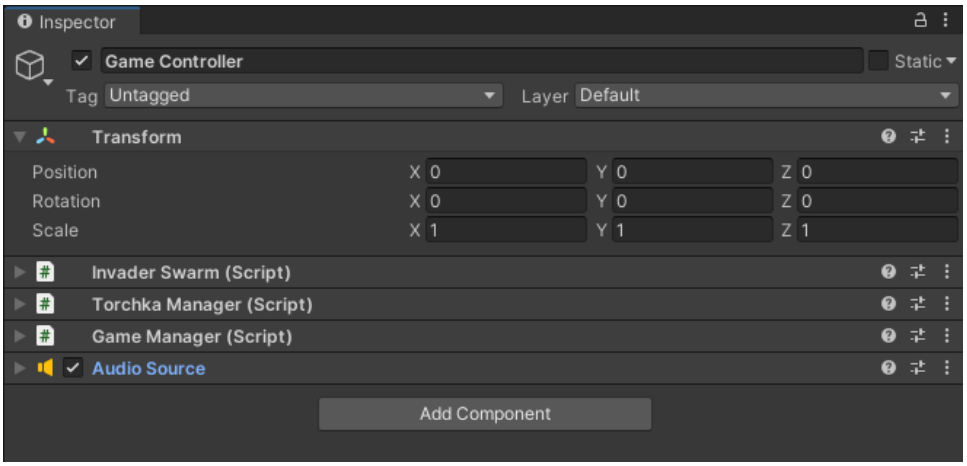


La detección de colisiones de Físicas 2D para los GameObjects en la capa Enemigo solo funciona con GameObjects en la capa Héroe. Los GameObjects de la capa Héroe funcionan tanto para Enemigo como para otros GameObjects en la capa Héroe. Esta información será importante más adelante.

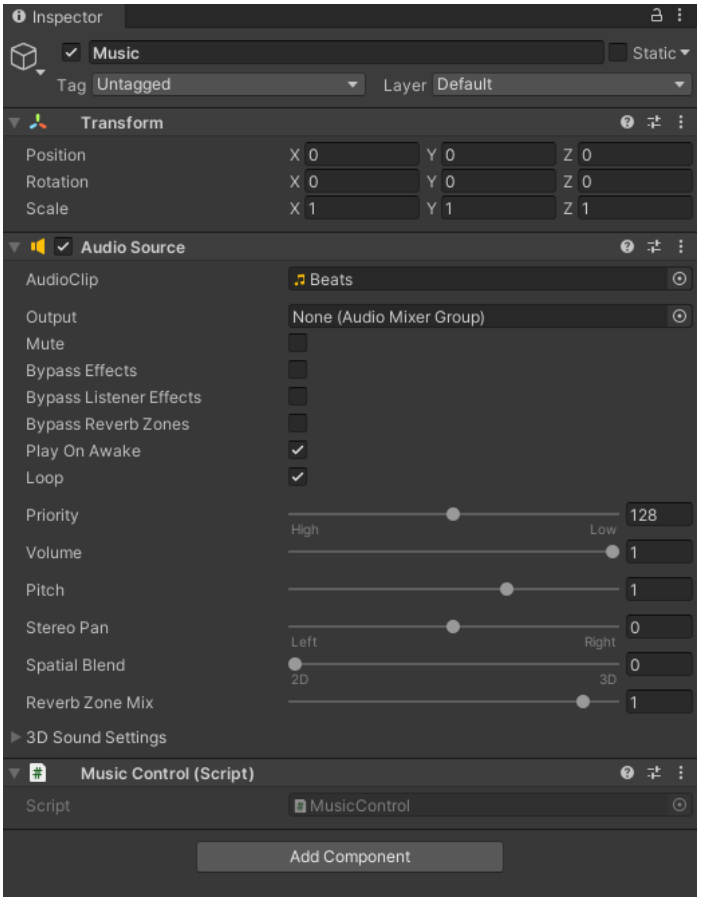
Ahora, echa un vistazo a los GameObjects en la Jerarquía. Algunos tienen componentes personalizados adjuntos en los que trabajarás en este tutorial.



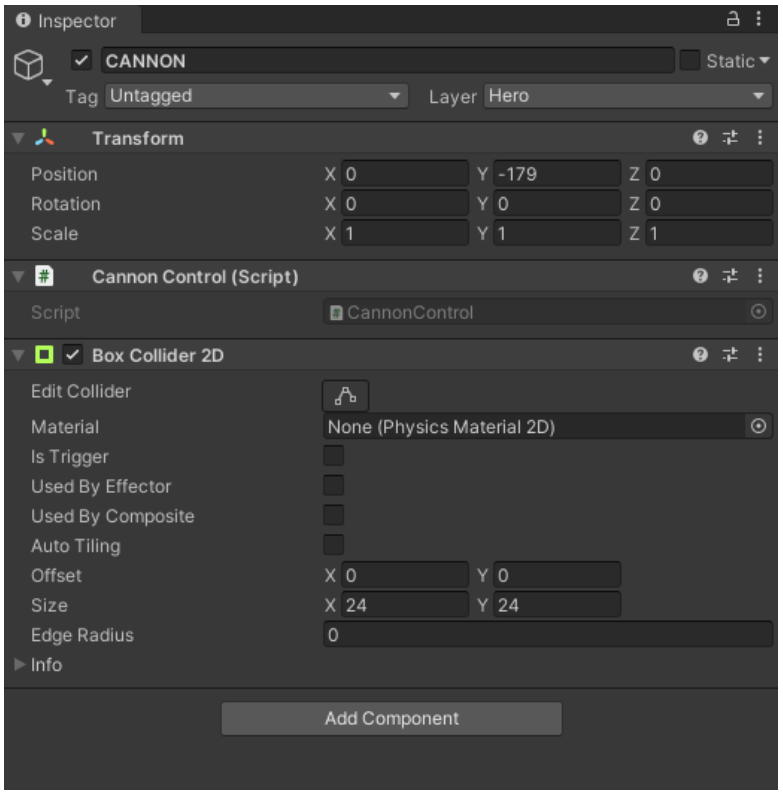
Game Controller tiene los componentes Invader Swarm, Torchka Manager y Game Manager adjuntos. Además de estos, hay un componente Audio Source que utilizarás para los efectos de sonido.



La música tiene Control de Música y una Fuente de Audio. La Fuente de Audio reproduce Beats ubicados en RW/Sonidos. Esta es la música principal del juego. Actualmente, la música tiene un tempo fijo, pero más adelante la harás dinámica.



CANNON tiene Control de Cañón y un Box Collider 2D para la detección de colisiones. Observa que su capa está configurada como Héroe. También tiene dos hijos inmediatos: Sprite, que es su sprite asociado, y Muzzle, un GameObject vacío que representa la posición para la instanciación de balas durante el disparo.



Main Camera es la única y principal cámara del juego y tiene el Audio Listener. Es ortográfica con un desplazamiento de posición de -10 a lo largo del eje Z.

Todos los demás GameObjects tienen Z establecido en 0. De hecho, para los demás, asume que el eje Z no existe ya que este es un juego en 2D, y tendremos que posicionarlos sólo en los ejes X e Y.

INVADERS Spawn Start y TORCHKA Spawn Center son dos GameObjects auxiliares vacíos que son hijos de Helpers. Sus posiciones te ayudarán a generar la nube de invasores y las torchkas más adelante.

Canvas y Event System son para la UI. Canvas tiene los siguientes hijos:

- **Texto de Puntuación:** Esto muestra la puntuación.
- **Texto de Vidas:** Utilizarás esto para mostrar las vidas restantes del jugador.
- **Panel de Game Over:** Este panel muestra el mensaje de Game Over cuando un jugador se queda sin vidas.
- **Todo Despejado:** Utilizarás este panel para mostrar el mensaje de Todo Despejado cuando el jugador elimine a todos los invasores.
- **Botón de Reinicio:** Este botón reinicia el juego.

Ahora que has terminado con el recorrido, es hora de agregar las cosas buenas. :]
En la siguiente sección, trabajarás en los controles del jugador.

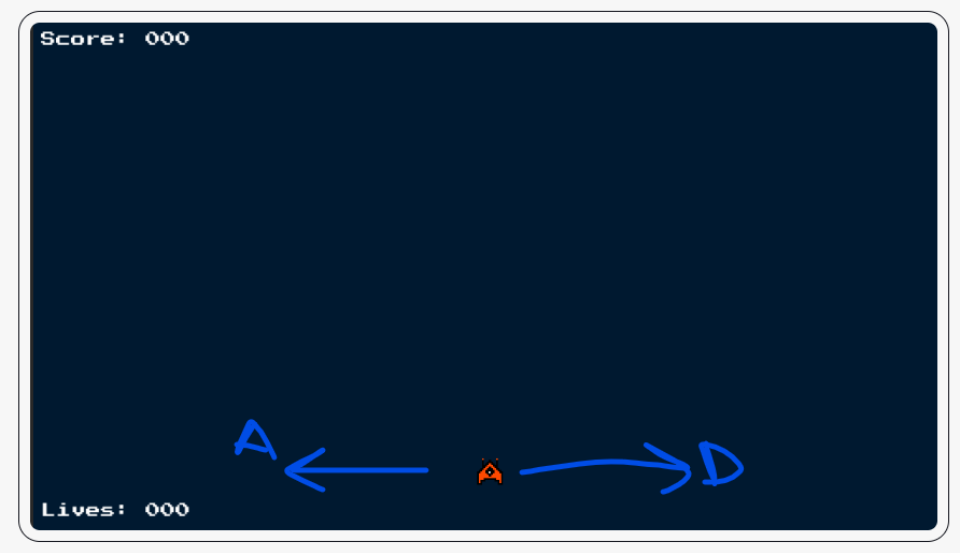
4- Creando los controles del Jugador

Selecciona el script CannonControl adjunto a CANNON. Ábrelo en tu editor de código. Pega el siguiente código dentro de la clase:

```
[SerializeField]
private float speed = 500f;

private void Update()
{
    if (Input.GetKey(KeyCode.D))
    {
        transform.Translate(speed * Time.deltaTime, 0, 0);
    }
    else if (Input.GetKey(KeyCode.A))
    {
        transform.Translate(-speed * Time.deltaTime, 0, 0);
    }
}
```

Aquí hay un desglose del código:

- La variable speed controla la velocidad del cañón.
- Dentro de Update, verificas si el jugador está manteniendo presionadas las teclas D o A. Si el jugador mantiene presionada D, el cañón se mueve a la derecha por speed * Time.deltaTime en cada fotograma. Si mantienen presionada la A, el cañón se mueve a la izquierda por la misma cantidad.
- Guarda el archivo. Regresa a Unity y selecciona Play dentro del editor. Ahora puedes mover la nave usando D y A.



A continuación, agregarás la mecánica de disparo utilizando el prefab Bullet dentro de RW/Prefabs.

Selecciona Bullet y echa un vistazo al Inspector.

El prefab tiene un componente Rigidbody 2D cinemático. Es cinemático porque no dependerás de la física para mover la bala, sino que la trasladarás mediante un script.

La capa del Rigidbody 2D está configurada como Héroe y tiene el componente Box Collider 2D para la detección de colisiones. También hay un componente Bullet que aún no hace nada.

Abre el script Bullet dentro de tu editor de código. Añade el siguiente código dentro de la clase:

```
[SerializeField]
private float speed = 200f;

[SerializeField]
private float lifeTime = 5f;

internal void DestroySelf()
{
    gameObject.SetActive(false);
    Destroy(gameObject);
}

private void Awake()
{
    Invoke("DestroySelf", lifeTime);
}

private void Update()
{
    transform.Translate(speed * Time.deltaTime * Vector2.up);
}

private void OnCollisionEnter2D(Collision2D other)
{
    DestroySelf();
}
```

En el código anterior, tú:

- **Usas Update** para mover la bala por speed * Time.deltaTime cada fotograma hacia la parte superior.
- **Usas DestroySelf** para destruir el GameObject de la bala. Antes de llamar a Destroy, desactivas la bala porque la destrucción tarda algunos fotogramas en procesarse, pero la desactivación es casi instantánea.
- **Awake invoca a DestroySelf** para destruir la bala automáticamente después de unos segundos definidos en lifetime. DestroySelf también se llama cuando una bala colisiona con otro GameObject.

Necesitarás efectos de sonido durante los disparos. Dentro de RW/Scripts, abre GameManager y añade lo siguiente dentro de la clase:

```
internal static GameManager Instance;

[SerializeField]
private AudioSource sfx;

internal void PlaySfx(AudioClip clip) => sfx.PlayOneShot(clip);

private void Awake()
{
    if (Instance == null)
    {
        Instance = this;
    }
    else if (Instance != this)
    {
        Destroy(gameObject);
    }
}
```

El código anterior convierte GameManager en un singleton, lo que asegura que solo tenga una única instancia en cualquier momento dado. También añade un método de utilidad, PlaySfx, que acepta un Audio Clip y lo reproduce utilizando la Fuente de Audio sfx.

Para usar la bala, necesitas instanciarla. Abre CannonControl de nuevo y añade lo siguiente después de la declaración de speed:

```
[SerializeField]
private Transform muzzle;

[SerializeField]
private AudioClip shooting;

[SerializeField]
private float coolDownTime = 0.5f;

[SerializeField]
private Bullet bulletPrefab;

private float shootTimer;
```

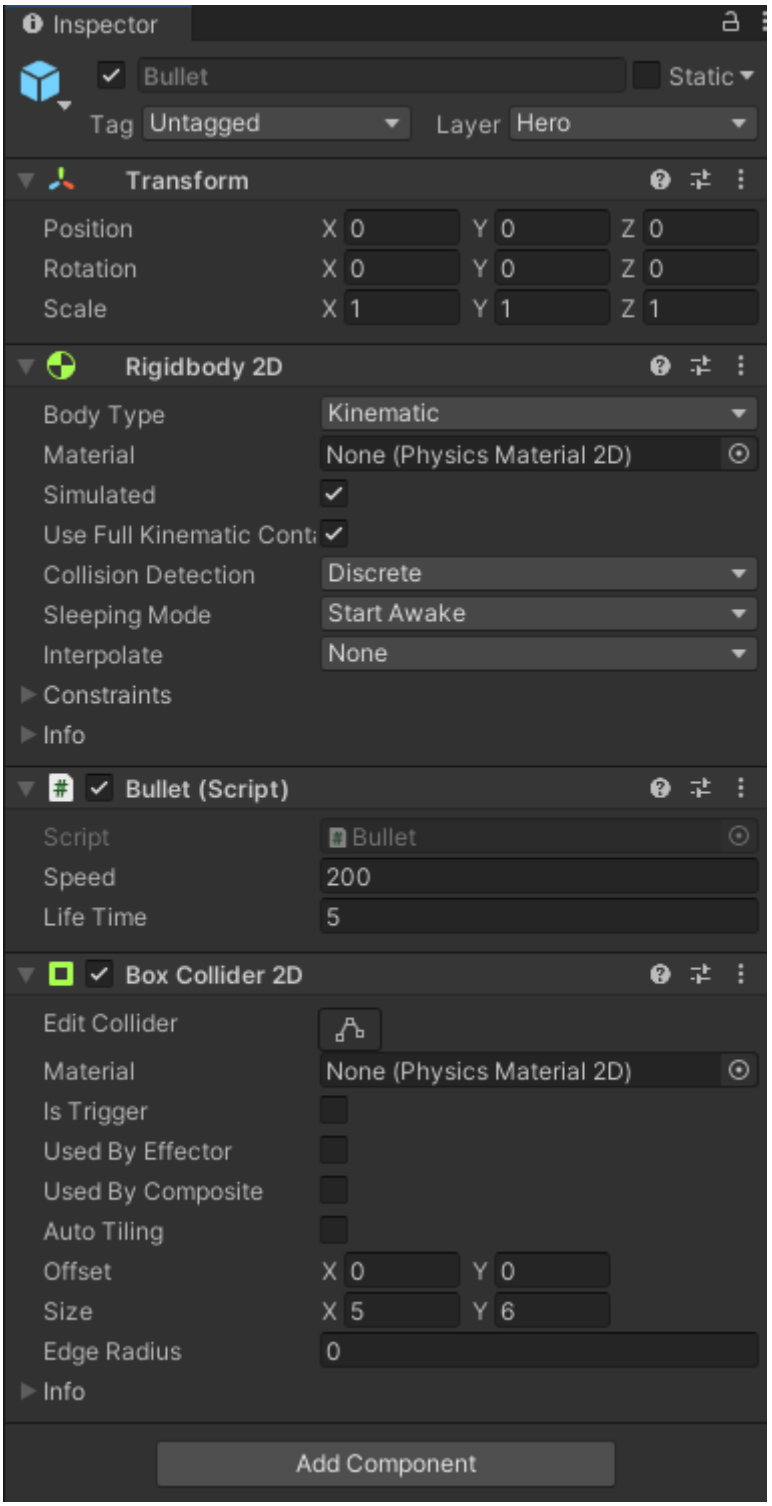
Entonces, después del código de movimiento anterior dentro de Update, pega:

```
shootTimer += Time.deltaTime;
if (shootTimer > coolDownTime && Input.GetKey(KeyCode.Space))
{
    shootTimer = 0f;

    Instantiate(bulletPrefab, muzzle.position,
Quaternion.identity);
    GameManager.Instance.PlaySfx(shooting);
}
```

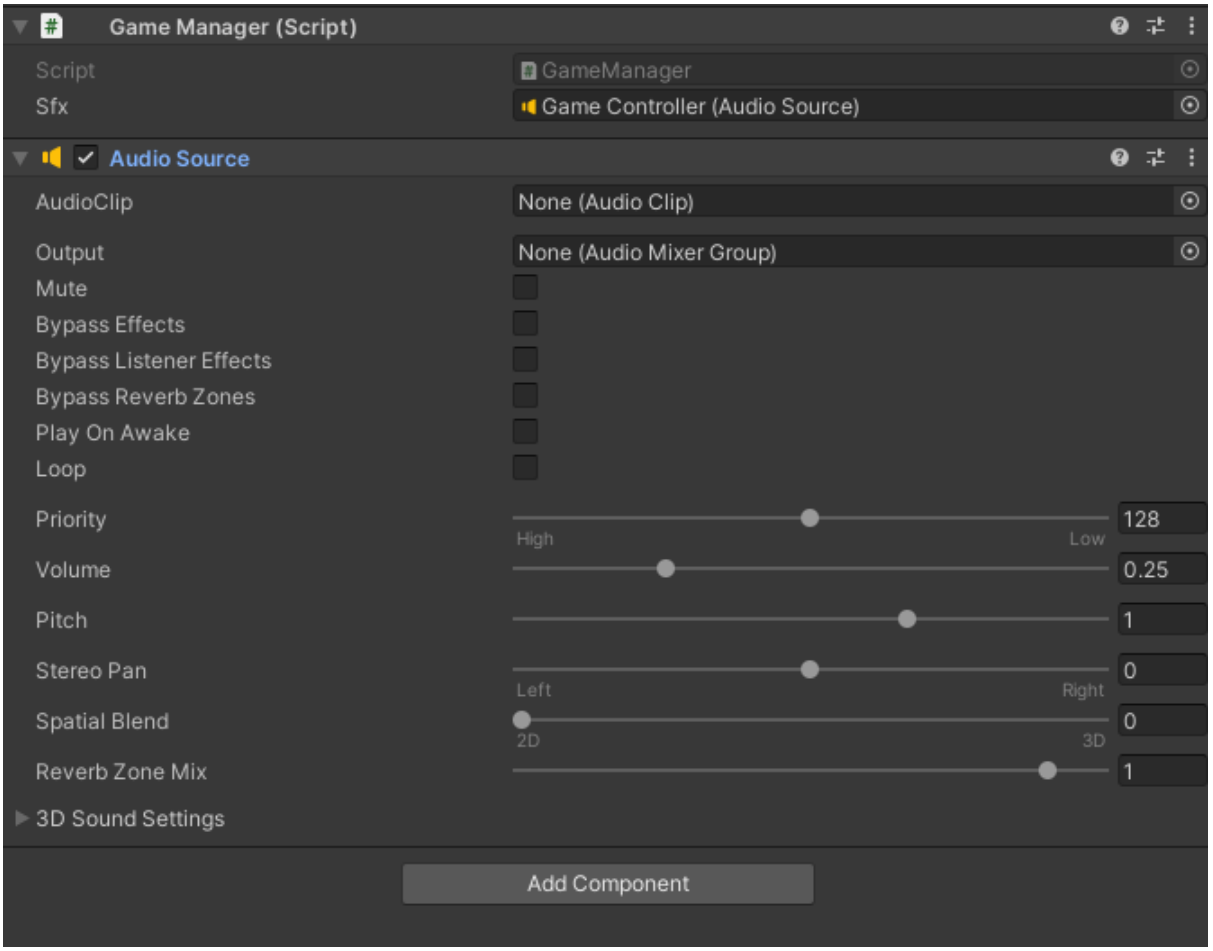
Este código incrementa shootTimer en cada fotograma hasta que alcanza coolDownTime. Si el jugador mantiene presionada la tecla Espacio, el código reinicia shootTimer e instancia bulletPrefab en muzzle.position. También reproduce el efecto de sonido de disparo utilizando PlaySfx dentro de GameManager.

Guarda todo y vuelve a Unity.

Regresa al prefab Bullet. Nota los nuevos campos para el componente Bullet.



Ahora, selecciona Game Controller desde la Hiercachy. Establece el Sfx de Game Manager en la Fuente de Audio adjunta al Game Controller.



Selecciona CANNON desde la Jerarquía. En Cannon Control, primero establece Muzzle en la Transformación Muzzle que es hija de CANNON. Luego, establece Bullet Prefab en la Bala que se encuentra en RW/Prefabs. Finalmente, establece Shooting en CannonBullet que se encuentra bajo RW/Sounds.

Guarda y dale a Play. Ahora puedes mover el cañón y disparar balas láser manteniendo presionada la tecla Espacio.

5- Creando los Invasores

¿Cómo empiezan todos los antagonistas sus vidas en los juegos? ¡Apareciendo, por supuesto! Haremos eso a continuación.

5.1 Haciendo Aparecer a los Invasores

Navega a RW/Scripts para examinar el script SimpleAnimator. Lo usarás para animar a los invasores, pero no necesitas preocuparte por su funcionamiento interno para este tutorial.

Cuando añades SimpleAnimator a un GameObject también añades un SpriteRenderer porque es un componente requerido para que el script funcione. SimpleAnimator también requiere un array de sprites. Seguirá actualizando el sprite de SpriteRenderer cíclicamente utilizando los sprites dentro del array, animándolo esencialmente.

Ahora, selecciona Invader Swarm adjunto a Game Controller. Abre el script asociado dentro de tu editor de código. Añade el siguiente código dentro de la clase:

```
 [System.Serializable]
private struct InvaderType
{
    public string name;
    public Sprite[] sprites;
    public int points;
    public int rowCount;
}
```

Aquí hay un desglose del código:

- Este código define una estructura InvaderType.
- El array sprites almacena todos los sprites asociados con el tipo de invasor, que SimpleAnimator usa para animar al invasor.
- rowCount es el número de filas que el tipo de invasor tendrá en la nube.
- name almacena el nombre del tipo para el invasor y points almacena la contribución a la puntuación si el jugador elimina un invasor de este tipo. Estos serán útiles más adelante.

Ahora añade lo siguiente justo debajo de InvaderType:

```
[Header("Spawning")]
[SerializeField]
private InvaderType[] invaderTypes;

[SerializeField]
private int columnCount = 11;

[SerializeField]
private int ySpacing;

[SerializeField]
private int xSpacing;

[SerializeField]
private Transform spawnStartPoint;

private float minX;
```

Utilizarás todos estos campos para la lógica de aparición:

- **invaderTypes**: Representa todos los tipos de invasores en uso.
- **columnCount**: Número total de columnas para la nube de invasores.
- **ySpacing**: El espacio entre cada invasor en la nube a lo largo del eje Y.
- **xSpacing**: El espacio entre cada invasor en la nube a lo largo del eje X.
- **spawnStartPoint**: Punto de aparición para el primer invasor.
- **minX**: Almacena el valor mínimo de la posición X para la nube.

A continuación, pega el siguiente código después de todas las declaraciones de variables:

```
private void Start()
{
    minX = spawnStartPoint.position.x;

    GameObject swarm = new GameObject { name = "Swarm" };
    Vector2 currentPos = spawnStartPoint.position;

    int rowIndex = 0;
    foreach (var invaderType in invaderTypes)
    {
        var invaderName = invaderType.name.Trim();
        for (int i = 0, len = invaderType.rowCount; i < len; i++)
        {
            for (int j = 0; j < columnCount; j++)
            {
                var invader = new GameObject() { name = invaderName };
                invader.AddComponent<SimpleAnimator>().sprites =
invaderType.sprites;
                invader.transform.position = currentPos;
                invader.transform.SetParent(swarm.transform);

                currentPos.x += xSpacing;
            }

            currentPos.x = minX;
            currentPos.y -= ySpacing;

            rowIndex++;
        }
    }
}
```

Aquí hay un desglose del código:

- Este código establece **minX** dentro de **Start**. Luego crea un GameObject vacío llamado Swarm.
- **xSpacing** e **ySpacing** actualizan **currentPos** a lo largo de los ejes X e Y, respectivamente.
- **currentPos.x** se incrementa después de cada iteración de invasor en la fila.
- Después de que se completa una fila, **currentPos.y** disminuye. Iterando sobre los miembros de **invaderTypes**, para cada invaderType, iteras fila por fila para crear GameObjects de invasores individuales en la posición currentPos.
- **xSpacing** e **ySpacing** actualizan **currentPos** a lo largo de los ejes X e Y, respectivamente.

- Cada GameObject de invasor creado tiene su nombre establecido en **invaderName**. Luego añades un componente SimpleAnimator, y asignas su array de sprites a los sprites asociados con el **invaderType**.
- Finalmente, el invasor se convierte en hijo de Swarm y su posición se establece en **currentPos**.
- Guarda todo y vuelve a Unity.

Selecciona Game Controller desde la Jerarquía. En Invader Swarm, establece:

- Y Spacing a 25
- X Spacing a 25
- Spawn Start Point a INVADERS Spawn Start, que es un hijo de Helpers.

Luego, establece el tamaño de Invader Types a 3 y establece los campos de los miembros de la siguiente manera:

0. Establece Name en SQUID, Points en 30 y Row Count en 1.
1. Establece Name en CRAB, Points en 20 y Row Count en 1.
2. Establece Name en OCTOPUS, Points en 10 y Row Count en 2.

Ahora, navega a RW/Sprites y mira la hoja de sprites de INVADERS. Estos no son los Space Invaders originales. Vuelve a Invader Swarm y configúralo de la siguiente manera, utilizando la hoja de sprites:

Para la entrada SQUID, establece la lista de Sprites para contener los siguientes sprites de la hoja de sprites en este orden:

- bugs_invaders_0
- bugs_invaders_5
- bugs_invaders_9
- bugs_invaders_4

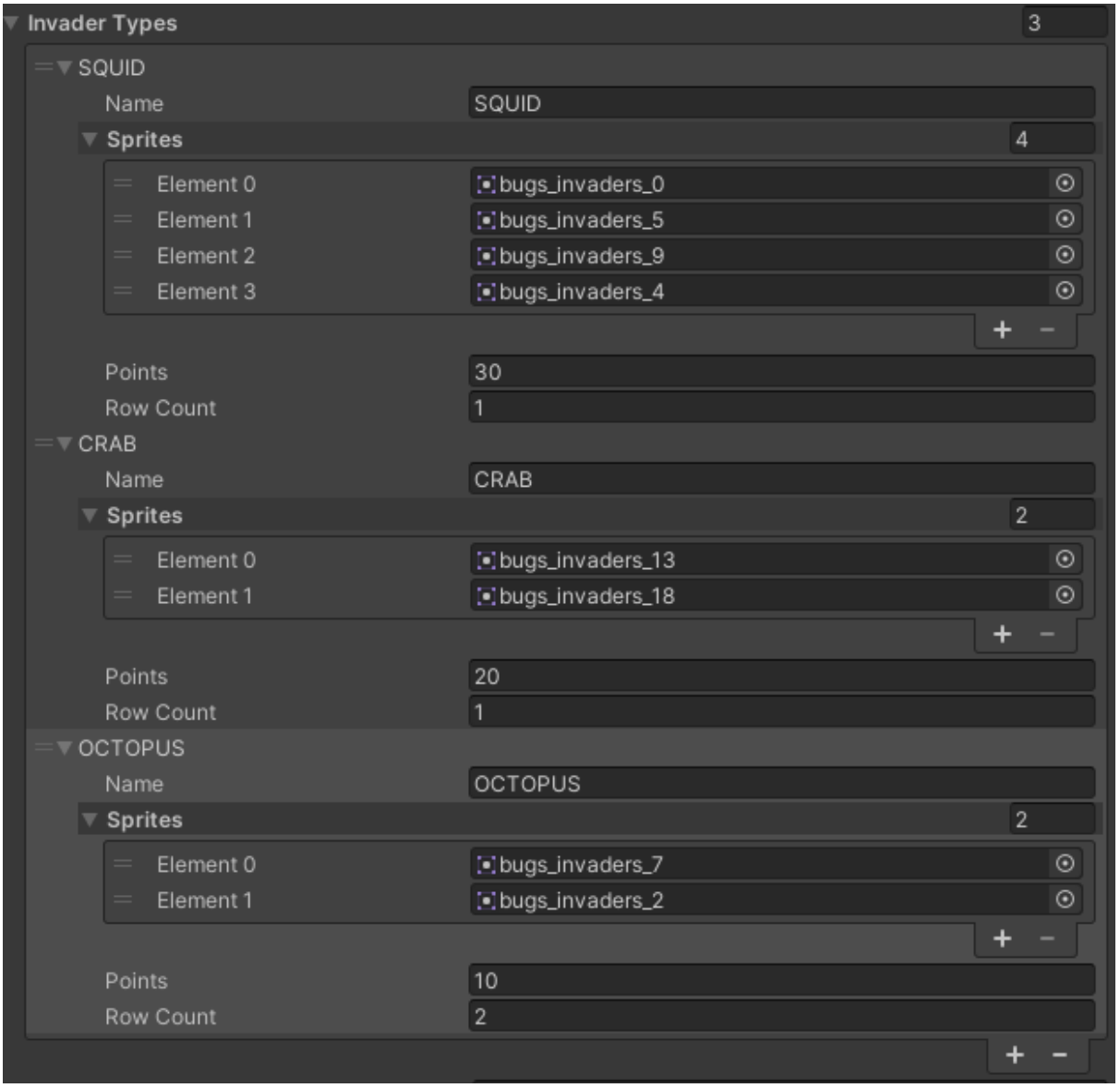
Realiza el mismo ejercicio, pero esta vez para CRAB usando los siguientes sprites:

- bugs_invaders_13
- bugs_invaders_18

Por último, asigna los sprites para OCTOPUS usando:

- bugs_invaders_7
- bugs_invaders_2

Aquí hay una referencia visual de cómo deberían verse las cosas ahora:



Guarda y dale a Play. Observa que la nube de invasores aparece y los invasores se animan en un lugar.

5.2 Mover a los Invasores

Regresa al script **InvaderSwarm.cs** y añade lo siguiente después de las declaraciones de variables existentes:

```
[Space]
[Header("Movement")]
[SerializeField]
private float speedFactor = 10f;

private Transform[,] invaders;
private int rowCount;
private bool isMovingRight = true;
private float maxX;
private float currentX;
private float xIncrement;
```

Todas estas variables ayudarán a mover la nube de invasores:
- **speedFactor** por ahora representa la velocidad a la que los invasores se mueven a lo largo del eje X. Más tarde, la velocidad involucrará el tempo de la música, por lo que la velocidad actual es el producto de los dos.

- **invaders** almacena las Transformaciones de todos los GameObjects de invasores creados.
- **rowCount** almacena el conteo total de filas de la nube.
- **isMovingRight** representa la dirección del movimiento y se establece en true por defecto.
- **maxX** es la posición X máxima para el movimiento de la nube.
- **currentX** representa la posición X general de la nube.
- **xIncrement** es el valor por fotograma que mueve a los invasores a lo largo del eje X.

Ahora, en Start, añade el siguiente código justo encima de int rowIndex = 0;

```
foreach (var invaderType in invaderTypes)
{
    rowCount += invaderType.rowCount;
}
maxX = minX + 2f * xSpacing * columnCount;
currentX = minX;
invaders = new Transform[rowCount, columnCount];
```

Este código calcula el conteo total de filas y lo almacena dentro de rowCount. Luego, calculas maxX basado en el total de columnas y el espaciado entre cada invasor. Inicialmente, currentX se establece en la posición X de spawnStartPoint.

Declaraste el array invaders. Para poblarlo, necesitarás una línea de código más. Pega la siguiente línea dentro del bucle for más interno, justo encima de currentPos.x += xSpacing;

```
invaders[rowIndex, j] = invader.transform;
```

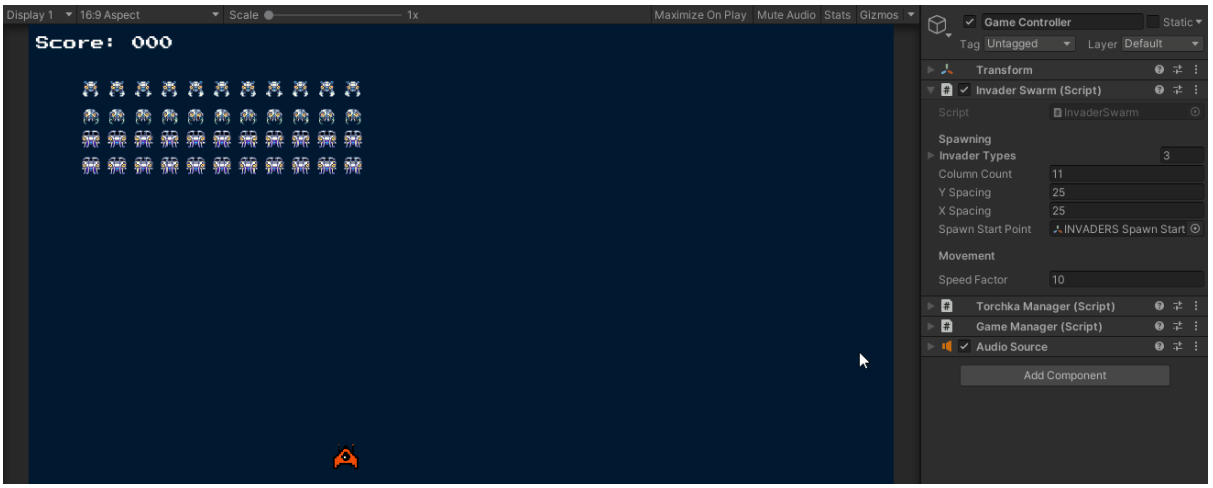
Esta línea se encarga de poblar invaders. Finalmente, justo después de Start, pega:

```
private void Update()
{
    xIncrement = speedFactor * Time.deltaTime;
    if (isMovingRight)
    {
        currentX += xIncrement;
        if (currentX < maxX)
        {
            MoveInvaders(xIncrement, 0);
        }
        else
        {
            ChangeDirection();
        }
    }
    else
    {
        currentX -= xIncrement;
        if (currentX > minX)
        {
            MoveInvaders(-xIncrement, 0);
        }
        else
        {
            ChangeDirection();
        }
    }
}

private void MoveInvaders(float x, float y)
{
    for (int i = 0; i < rowCount; i++)
    {
        for (int j = 0; j < columnCount; j++)
        {
            invaders[i, j].Translate(x, y, 0);
        }
    }
}

private void ChangeDirection()
{
    isMovingRight = !isMovingRight;
    MoveInvaders(0, -ySpacing);
}
```

Desglose del código:

- **MoveInvaders** acepta dos valores float: x e y. Mueve cada Transform en invaders por el mismo valor a lo largo de los ejes X e Y respectivamente.
- **ChangeDirection** alterna isMovingRight y mueve la nube hacia abajo por la cantidad de ySpacing.
- **Dentro de Update**, calculas xIncrement y actualizas currentX basándote en la dirección en cada fotograma.
- Usas **currentX** para verificar si la posición X de la nube se está acercando a un umbral. Si es así, llamas a ChangeDirection. Si no, mueves la nube usando MoveInvaders.

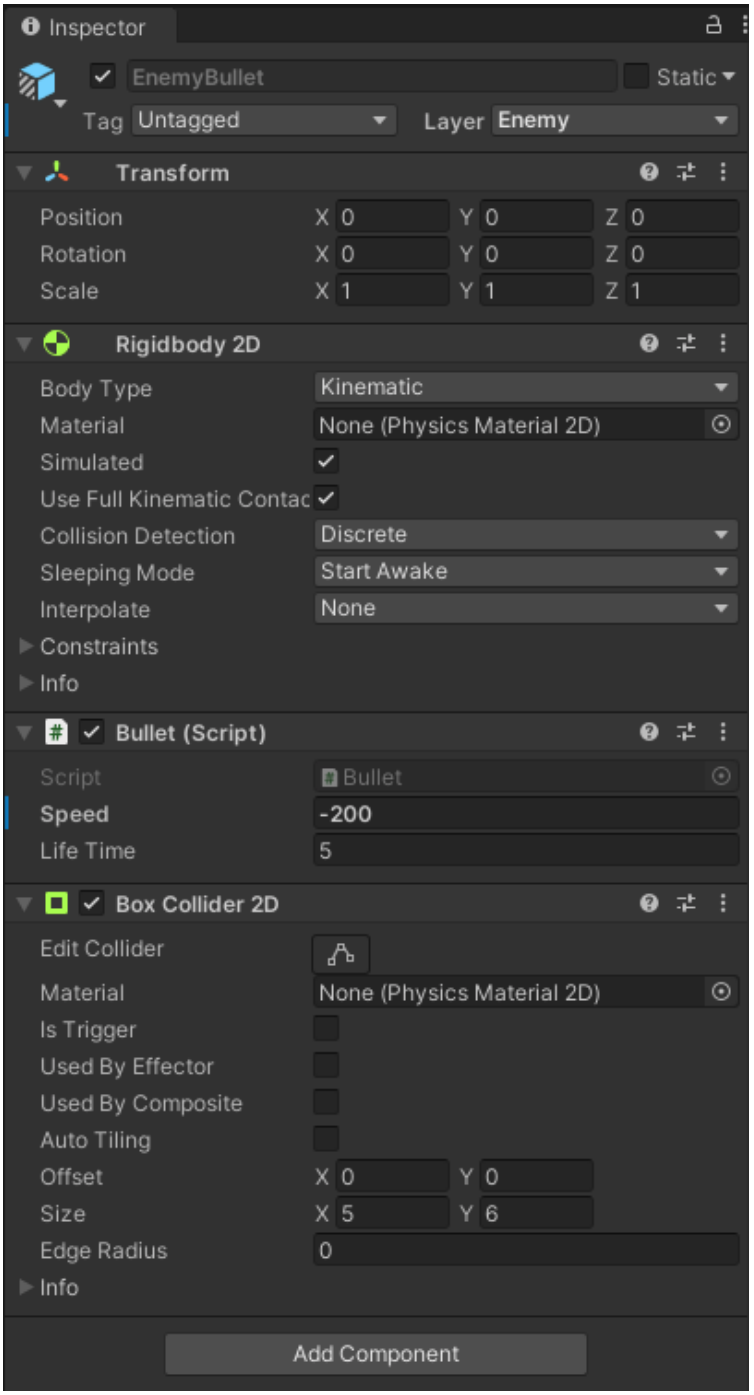
Guarda todo. Vuelve a Unity y haz clic en Play. Verás la nube de invasores moviéndose. Mientras estás en el Modo Play, prueba diferentes valores para el Factor de Velocidad de la Nube de Invasores y observa cómo afecta a la velocidad de la nube.



5.3 Disparos de los Invasores

Utilizarás una variante del prefab Bullet para los invasores. Navega a RW/Prefabs y encuentra el EnemyBullet. Es igual que Bullet, excepto que el sprite apunta en la dirección Y opuesta y su capa está configurada como Enemigo.

Selecciona EnemyBullet. Nota que la Velocidad de Bullet está establecida en -200. Esto asegura que la bala se mueva en la dirección opuesta a las balas del cañón pero con la misma magnitud.



En el juego, solo los invasores en el frente de la nube disparan balas láser. Para lograr esto, utilizarás el prefab BulletSpawner ubicado en RW/Prefabs.

Instanciarás tantos de ellos como la cantidad de columnas de invasores. También harás que los spawners de balas sigan las Transformaciones de los invasores en el frente de la nube. Esto asegurará que, al disparar, las balas parecerán provenir de los invasores de la fila frontal.

Antes de hacer eso, necesitas una manera de obtener una Transformación de invasor en una fila y columna específicas desde el array de invasores dentro de InvaderSwarm.

Abre el script **InvaderSwarm.cs** y añade la siguiente línea después de la declaración de la estructura InvaderType:

```
internal static InvaderSwarm Instance;
```

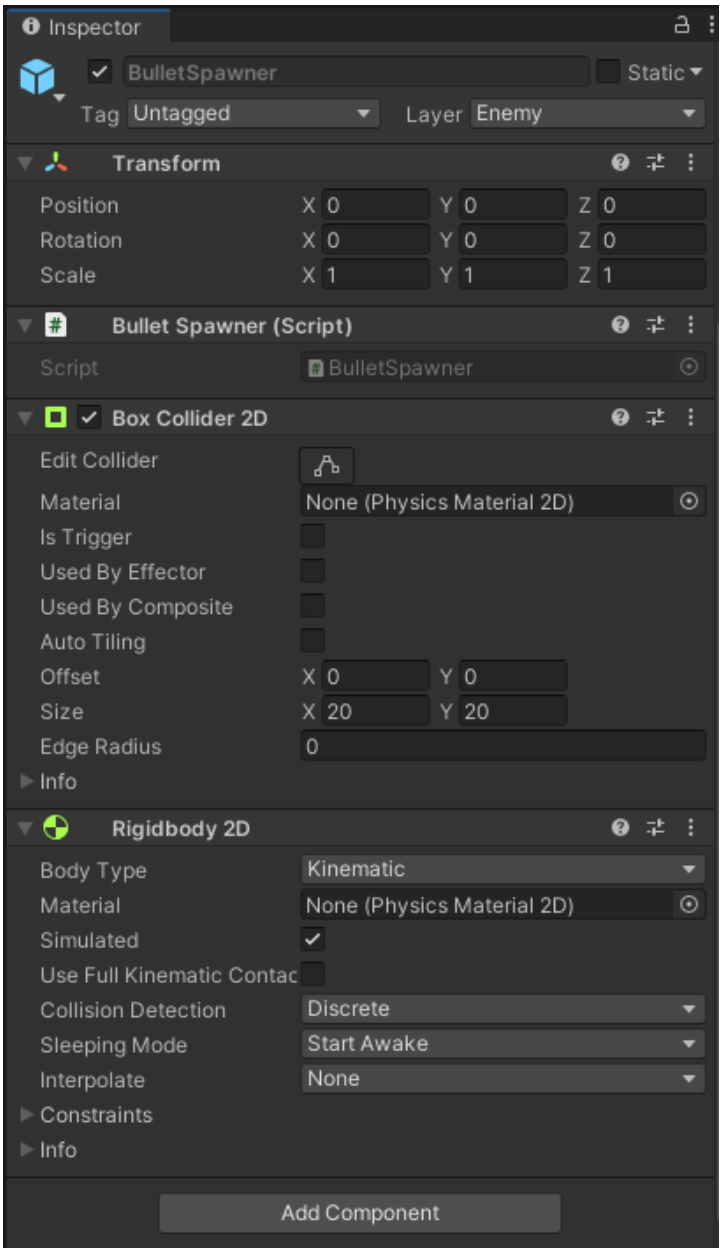
Esto ayuda a convertir el InvaderSwarm en un Singleton. Luego pega lo siguiente justo encima de Start:

```
internal Transform GetInvader(int row, int column)
{
    if (row < 0 || column < 0
        || row >= invaders.GetLength(0) || column >=
invaders.GetLength(1))
    {
        return null;
    }

    return invaders[row, column];
}

private void Awake()
{
    if (Instance == null)
    {
        Instance = this;
    }
    else if (Instance != this)
    {
        Destroy(gameObject);
    }
}
```

GetInvader devuelve la Transformación del invasor en el índice de fila y columna de invaders. Awake convierte el InvaderSwarm en un Singleton asegurando que, cuando el juego comienza, solo una instancia de InvaderSwarm estará activa.

Ahora, selecciona el prefab BulletSpawner y echa un vistazo al Inspector. Nota que tiene Bullet Spawner adjunto a él.

También hay un Rigidbody 2D cinemático, un Box Collider 2D y la capa está configurada como Enemigo. No añadirás colisionadores a los invasores, sino que usarás este colisionador para detectar impactos de las balas del cañón.



En tu editor de código, abre el script BulletSpawner.cs adjunto a BulletSpawner y añade lo siguiente dentro de la clase:

```
internal int currentRow;
internal int column;

[SerializeField]
private AudioClip shooting;

[SerializeField]
private GameObject bulletPrefab;

[SerializeField]
private Transform spawnPoint;

[SerializeField]
private float minTime;

[SerializeField]
private float maxTime;

private float timer;
private float currentTime;
private Transform followTarget;

internal void Setup()
{
    currentTime = Random.Range(minTime, maxTime);
    followTarget = InvaderSwarm.Instance.GetInvader(currentRow, column);
}

private void Update()
{
    transform.position = followTarget.position;

    timer += Time.deltaTime;
    if (timer < currentTime)
    {
        return;
    }

    Instantiate(bulletPrefab, spawnPoint.position, Quaternion.identity);
    GameManager.Instance.PlaySfx(shooting);
    timer = 0f;
    currentTime = Random.Range(minTime, maxTime);
}
```

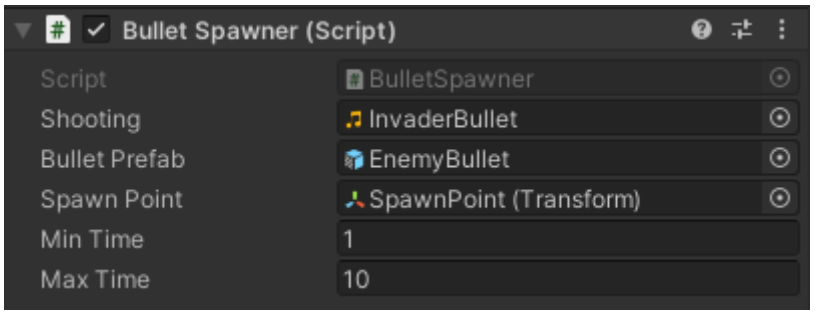
Desglose del código:

- **currentTime** representa el tiempo a esperar hasta disparar la siguiente bala. Se establece a un valor aleatorio entre minTime y maxTime.
- **currentRow** y column vinculan un spawner de balas con un invasor. La columna se establece una vez y no cambiará. Pero, como verás más adelante, currentRow se actualiza si las balas del jugador golpean este spawner.
- **Dentro de Setup()**, estableces followTarget llamando a GetInvader desde la instancia de InvaderSwarm usando currentRow y column. También estableces un valor inicial para currentTime.

Dentro de Update, actualizas la posición del spawner de balas para que coincida con followTarget.position. Además, incrementas el temporizador hasta que alcanza currentTime. Cuando esto sucede, creas una bala en spawnPoint.position mientras se reproduce el efecto de sonido de disparo, seguido de reiniciar el temporizador y currentTime.

Guarda todo. Regresa a Unity y abre BulletSpawner en Modo Prefab. Asegúrate de que los siguientes valores estén establecidos para Bullet Spawner:

- Shooting a InvaderBullet ubicado en RW/Sounds.
- Bullet Prefab a EnemyBullet ubicado en RW/Prefabs.
- Spawn Point al Transform SpawnPoint que es el único hijo de BulletSpawner.
  Min Time a 1 y Max Time a 10.



Para usar el BulletSpawner, necesitas volver al script InvaderSwarm.cs. Primero, pega la siguiente línea al final de todas las declaraciones de variables:

```
[SerializeField]
private BulletSpawner bulletSpawnerPrefab;
```

Dentro de Start añadimos esta línea al final:

```
for (int i = 0; i < columnCount; i++)
{
    var bulletSpawner = Instantiate(bulletSpawnerPrefab);
    bulletSpawner.transform.SetParent(swarm.transform);
    bulletSpawner.column = i;
    bulletSpawner.currentRow = rowCount - 1;
    bulletSpawner.Setup();
}
```

En este código, creas un spawner de balas y lo configuras. Instancias bulletSpawner para cada columna de la nube y estableces su column y currentRow, seguido de la llamada a su método Setup. También haces que bulletSpawner sea hijo de Swarm para evitar desorden en la Jerarquía.

Guarda todo y vuelve a Unity. Selecciona Game Controller desde la Jerarquía y establece el Prefab de Bullet Spawner para Invader Swarm en BulletSpawner, ubicado en RW/Prefabs.

Guarda y juega. Ahora tienes invasores que disparan balas al jugador.

Observa que ambas balas desaparecen cuando una bala de invasor y una bala de cañón colisionan. La bala del invasor desaparece cuando golpea al cañón y la bala del cañón desaparece cuando golpea a un spawner de balas. Desaparecen porque OnCollisionEnter2D llama a DestroySelf dentro de Bullet.

Sin embargo, falta una cosa, y esas son las explosiones 🙂 Las añadiremos a continuación.

5.4 Añadiendo las Explosiones

Navega a RW/Prefabs. Observa el prefab Explosion. Es un GameObject simple con una animación de sprite prehecha que usarás para las visuales de la explosión.

Ahora, abre **GameManager.cs** de nuevo. Añade lo siguiente después de declarar sfx:

```
[SerializeField]
private GameObject explosionPrefab;

[SerializeField]
private float explosionTime = 1f;

[SerializeField]
private AudioClip explosionClip;

internal void CreateExplosion(Vector2 position)
{
    PlaySfx(explosionClip);

    var explosion = Instantiate(explosionPrefab, position,
        Quaternion.Euler(0f, 0f, Random.Range(-180f, 180f)));
    Destroy(explosion, explosionTime);
}
```

CreateExplosion crea una explosión en position con una rotación aleatoria a lo largo del eje Z, y la destruye después de explosionTime segundos.

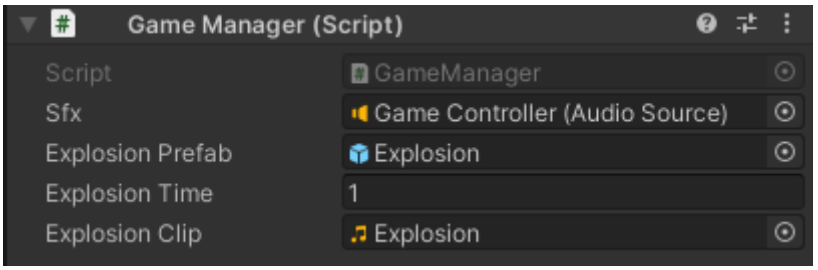
Para usar CreateExplosion, añade la siguiente línea al final de DestroySelf dentro de la clase Bullet:

```
GameManager.Instance.CreateExplosion(transform.position);
```

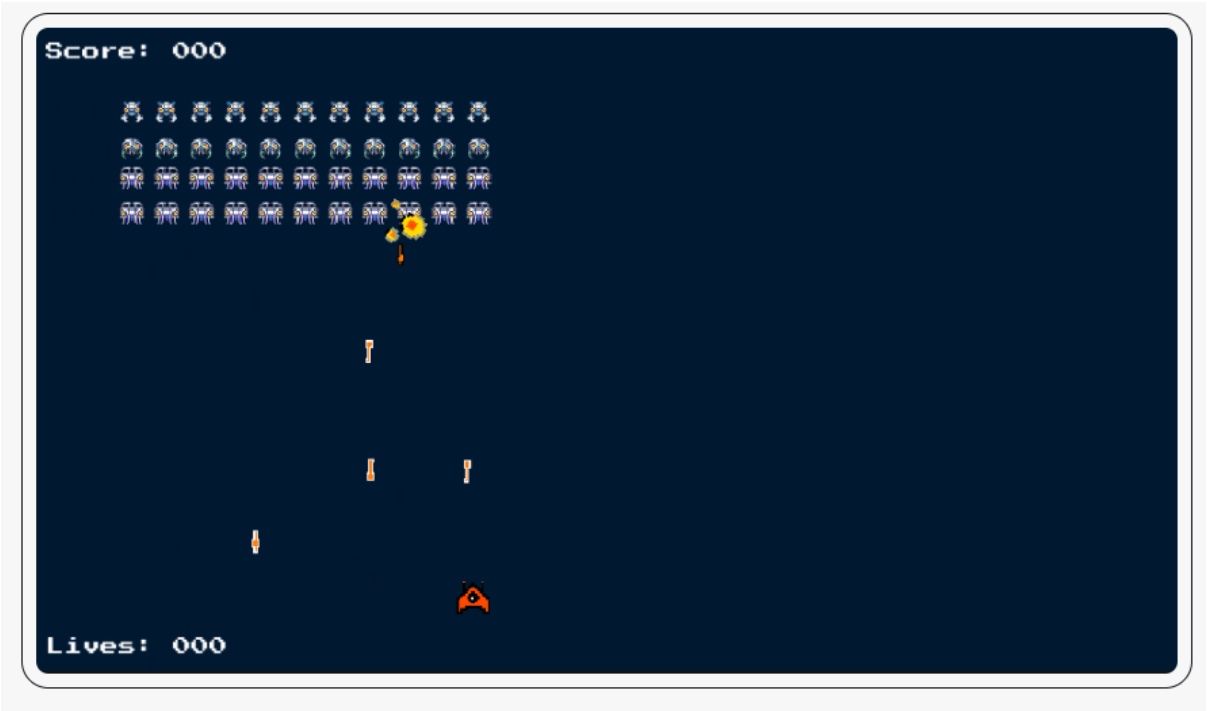
Guarda todo. Regresa a Unity y selecciona Game Controller desde la Jerarquía.

Para el Game Manager establece:

- Explosion Prefab a Explosion ubicado en RW/Prefabs.
- Explosion Clip a Explosion ubicado en RW/Sounds.



Guardamos, damos al play y ahora tenemos explosiones



6- Añadir las Vidas y el Marcador de Juego

6.1 Vidas del Jugador

Abre GameManager.cs y añade el siguiente código después de las variables:

```
[SerializeField]
private int maxLives = 3;

[SerializeField]
private Text livesLabel;

private int lives;

internal void UpdateLives()
{
    lives = Mathf.Clamp(lives - 1, 0, maxLives);
    livesLabel.text = $"Lives: {lives}";
}
```

Llamar a UpdateLives decrementa la variable lives en uno y actualiza la etiqueta de la UI para reflejar el cambio. Actualmente, no sucede nada cuando lives llega a cero, pero cambiarás eso más adelante. Añade lo siguiente al final de Awake:

```
lives = maxLives;
livesLabel.text = $"Lives: {lives}";
```

Este código establece el valor predeterminado para lives y también actualiza la etiqueta de la UI.

Ahora, abre CannonControl y pega las siguientes líneas después de las declaraciones de variables:

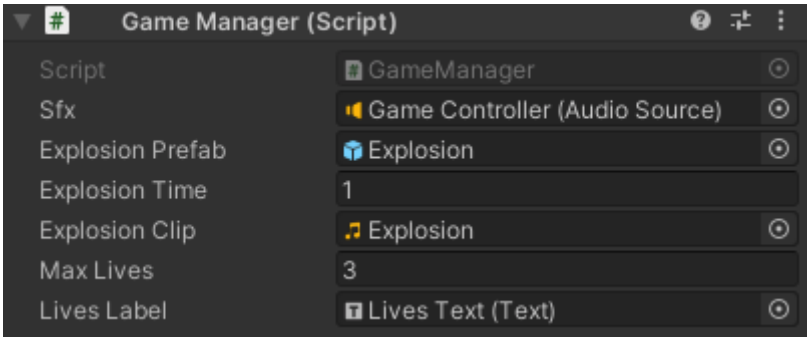
```
[SerializeField]
private float respawnTime = 2f;

[SerializeField]
private SpriteRenderer sprite;

[SerializeField]
private Collider2D cannonCollider;

private Vector2 startPos;

private void Start() => startPos = transform.position;
```

Añadimos este código después de Update:

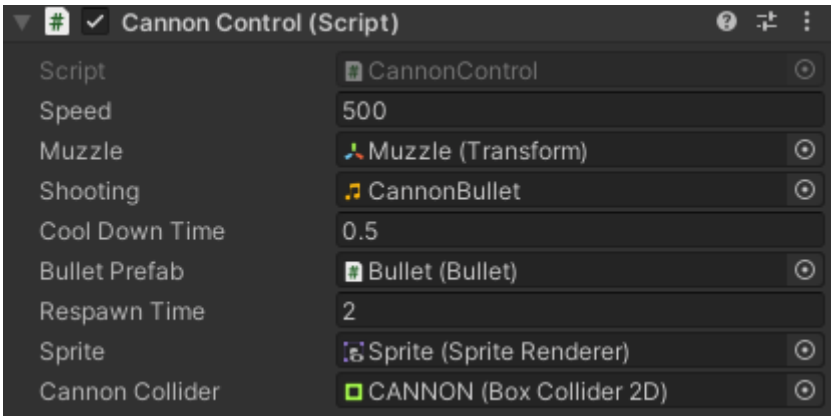
```
private void OnCollisionEnter2D(Collision2D other)
{
    GameManager.Instance.UpdateLives();
    StopAllCoroutines();
    StartCoroutine(Respawn());
}

System.Collections.IEnumerator Respawn()
{
    enabled = false;
    cannonCollider.enabled = false;
    ChangeSpriteAlpha(0.0f);

    yield return new WaitForSeconds(0.25f * respawnTime);

    transform.position = startPos;
    enabled = true;
    ChangeSpriteAlpha(0.25f);

    yield return new WaitForSeconds(0.75f * respawnTime);

    ChangeSpriteAlpha(1.0f);
    cannonCollider.enabled = true;
}

private void ChangeSpriteAlpha(float value)
{
    var color = sprite.color;
    color.a = value;
    sprite.color = color;
}
```

Esto es lo que está sucediendo:

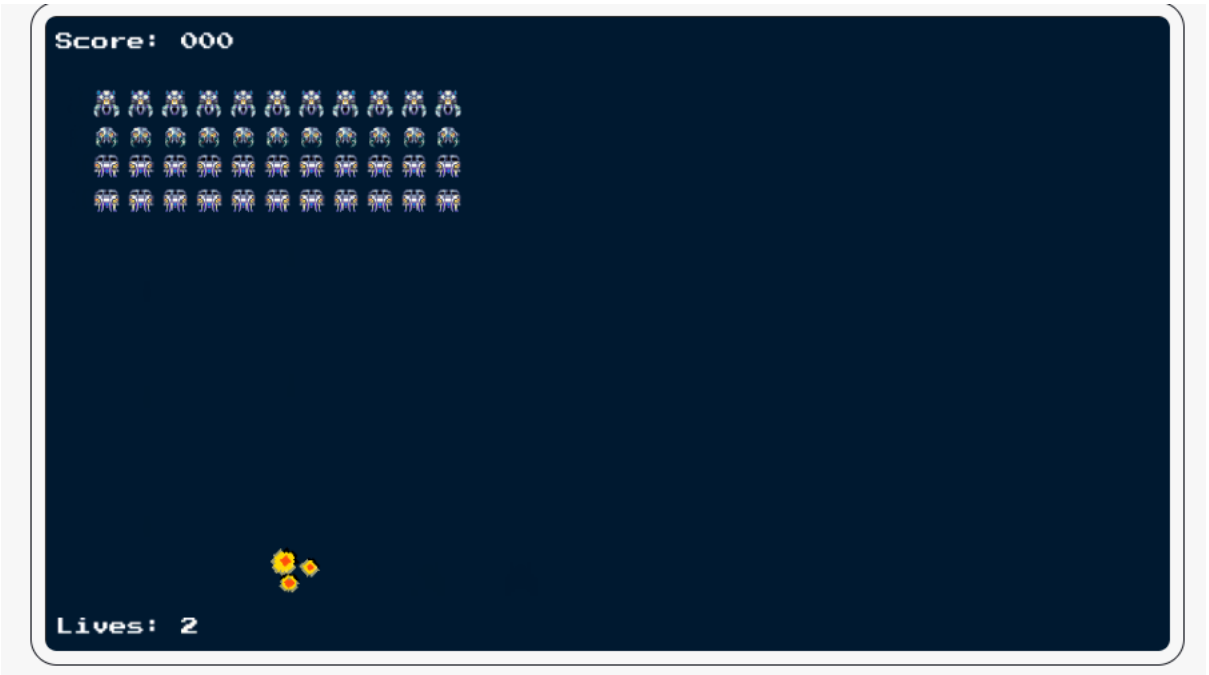
- ChangeSpriteAlpha cambia la opacidad del sprite del cañón.
- Cuando una bala golpea el cañón, GameManager.UpdateLives decrementa el total de vidas y comienza la corrutina Respawn.
- Respawn primero desactiva cannonCollider y hace invisible el sprite del cañón. Después de unos momentos, hace que el sprite del cañón sea ligeramente transparente y establece la posición del cañón de nuevo a startPos. Finalmente, restaura la opacidad del sprite y vuelve a activar el colisionador.
- Guarda todo y vuelve a Unity. Selecciona Game Controller y establece la Etiqueta de Vidas del Game Manager a Lives Text, que es hijo de Canvas.



Para el Cannon Control en CANNON, establece Sprite al Sprite Renderer en Sprite, un GameObject hijo de CANNON. Establece Collider al Box Collider 2D en el CANNON.



Ahora los invasores nos pueden matar



6.2 Marcador de Juego y Game Over

Antes de que hagas cualquier otra cosa, abre el script **MusicControl.cs**. Quieres detener la música cuando el juego termine, así que pega el siguiente código dentro de la clase:

```
[SerializeField]
private AudioSource source;

internal void StopPlaying() => source.Stop();
```

StopPlaying detiene la fuente de audio cuando se llama.

Ahora, abre el script GameManager.cs y añade lo siguiente después de las declaraciones de variables:

```
[SerializeField]
private MusicControl music;

[SerializeField]
private Text scoreLabel;

[SerializeField]
private GameObject gameOver;

[SerializeField]
private GameObject allClear;

[SerializeField]
private Button restartButton;

private int score;

internal void UpdateScore(int value)
{
    score += value;
    scoreLabel.text = $"Score: {score}";
}

internal void TriggerGameOver(bool failure = true)
{
    gameOver.SetActive(failure);
    allClear.SetActive(!failure);
    restartButton.gameObject.SetActive(true);

    Time.timeScale = 0f;
    music.StopPlaying();
}
```

Luego, pega las siguientes líneas al final de UpdateLives:

```
if (lives > 0)
{
    return;
}

TriggerGameOver();
```

Finalmente, añade lo siguiente al final de Awake:

```
score = 0;
scoreLabel.text = $"Score: {score}";
gameOver.gameObject.SetActive(false);
allClear.gameObject.SetActive(false);

restartButton.onClick.AddListener(() =>
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    Time.timeScale = 1f;
});
restartButton.gameObject.SetActive(false);
```

Esto es lo que hace este código:

- allClear almacena una referencia al panel "All Clear" (Todo Despejado), que se muestra cuando el jugador elimina a todos los invasores. gameOver hace referencia al panel "Game Over" (Fin del Juego) que se muestra cuando el jugador se queda sin vidas.
- UpdateScore incrementa score por el valor que se le pasa y actualiza la etiqueta de la UI para reflejar los cambios.
- TriggerGameOver muestra el panel "Game Over" si failure es verdadero. De lo contrario, muestra el panel "All Clear". También habilita el restartButton, pausa el juego y detiene la música.
- Awake maneja el evento onClick para el botón de reinicio. Recarga la escena cuando se hace clic.

Abre InvaderSwarm.cs y añade lo siguiente dentro de la clase después de las declaraciones de variables:

```
private int killCount;
private System.Collections.Generic.Dictionary<string, int> pointsMap;

internal void IncreaseDeathCount()
{
    killCount++;
    if (killCount >= invaders.Length)
    {
        GameManager.Instance.TriggerGameOver(false);
        return;
    }
}

internal int GetPoints(string alienName)
{
    if (pointsMap.ContainsKey(alienName))
    {
        return pointsMap[alienName];
    }
    return 0;
}
```

Luego pega la siguiente línea justo arriba de int rowIndex = 0; dentro de Start:

```
pointsMap = new System.Collections.Generic.Dictionary<string, int>();
```

Debajo de la línea, justo debajo de var invaderName = invaderType.name.Trim(); añade lo siguiente:

```
pointsMap[invaderName] = invaderType.points;
```

Desglose del código:

- pointsMap es un Diccionario (un mapa de string a entero). Mapea el nombre del tipo de invasor con su valor de puntos.
- IncreaseDeathCount lleva la cuenta y actualiza killCount cuando el jugador elimina un invasor. Cuando el jugador elimina a todos los invasores, TriggerGameOver recibe falso y muestra el panel "All Clear" (Todo Despejado).
- GetPoints devuelve los puntos asociados con un tipo de invasor pasando su nombre como clave.

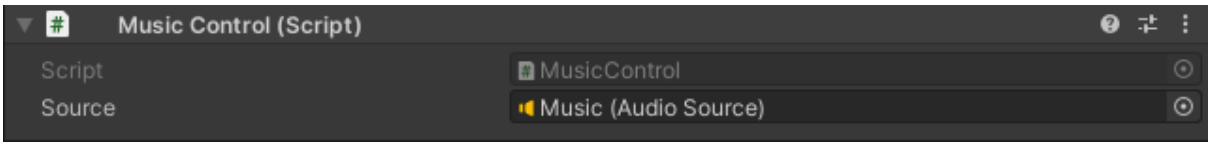
Finalmente, abre BulletSpawner.cs para manejar la detección de colisiones para los invasores. Pega lo siguiente justo después de Update:

```
private void OnCollisionEnter2D(Collision2D other)
{
    if (!other.collider.GetComponent<Bullet>())
    {
        return;
    }

    GameManager.Instance.

UpdateScore(InvaderSwarm.Instance.GetPoints(followTarget.gameObject.name))
;

    InvaderSwarm.Instance.IncreaseDeathCount();

    followTarget.GetComponentInChildren<SpriteRenderer>().enabled = false;
    currentRow = currentRow - 1;
    if (currentRow < 0)
    {
        gameObject.SetActive(false);
    }
    else
    {
        Setup();
    }
}
```

Esto es lo que hace este código:

- OnCollisionEnter2D retorna sin hacer nada si el objeto que golpeó el spawner de la bala no era del tipo Bullet.
- Si la bala del cañón golpea el spawner, la puntuación y la cuenta de muertes se actualizan. Además, el Sprite Renderer del followTarget actual se desactiva, luego actualiza el currentRow.
- Si no quedan filas, el GameObject se desactiva. De lo contrario, llamas a Setup para actualizar el followTarget.

Selecciona Music y establece Source de Music Control al componente Audio Source en el mismo GameObject



Luego, selecciona Game Controller. Para Game Manager, establece:

- Music a Music Control de Music.
- Score Label a Score Text.
- Game Over a Game Over Panel.
- All Clear a All Clear.
- Restart Button a Restart Button.

Mata a todos los invasores para ver el mensaje "All Clear" (Todo Despejado), o deja que las vidas se agoten para ver el mensaje "Game Over" (Fin del Juego). Luego, puedes usar el botón "Restart" (Reiniciar) para recargar la escena.

6.3 Añadir el Audio Dinámico

Ve a MusicControl.cs. Ábrelo en tu editor de código. Añade la siguiente línea en la parte superior de la clase.

```
private readonly float defaultTempo = 1.33f;
```

Esta línea representa los golpes por segundo predeterminados de la música. Puedes calcular este valor considerando que la música tiene cuatro golpes y dura tres segundos.

Ahora, pega lo siguiente arriba de StopPlaying:

```
[SerializeField]
internal int pitchChangeSteps = 5;

[SerializeField]
private float maxPitch = 5.25f;

private float pitchChange;

internal float Tempo { get; private set; }
```

Luego, añade las siguientes líneas después de la definición de StopPlaying:

```
internal void IncreasePitch()
{
    if (source.pitch == maxPitch)
    {
        return;
    }

    source.pitch = Mathf.Clamp(source.pitch + pitchChange, 1, maxPitch);
    Tempo = Mathf.Pow(2, pitchChange) * Tempo;
}

private void Start()
{
    source.pitch = 1f;
    Tempo = defaultTempo;
    pitchChange = maxPitch / pitchChangeSteps;
}
```

Así es como funciona el código:

- Dentro de Start, source.pitch y Tempo se establecen en sus valores predeterminados.
- IncreasePitch incrementa el tono del audio de source por una cantidad dictada por pitchChange, que a su vez es la relación entre maxPitch y pitchChangeSteps. maxPitch también pone un límite superior al tono.
- Después de cambiar el tono, puedes calcular el tempo basado en la siguiente fórmula:

$$pitchchange = -\log_2(tempo_1/tempo_2)$$

where $tempo_1$ denotes the tempo before the change and $tempo_2$ the tempo after the change.

Ahora abre el script InvaderSwarm.cs y añade lo siguiente al final de las declaraciones de variables:

```
[SerializeField]
private MusicControl musicControl;

private int tempKillCount;
```

En IncreaseDeathCount, pega las siguientes líneas al final:

```
tempKillCount++;
if (tempKillCount < invaders.Length / musicControl.pitchChangeSteps)
{
    return;
}

musicControl.IncreasePitch();
tempKillCount = 0;
```

Ahora IncreaseDeathCount rastrea la variable tempKillCount para verificar si supera invaders.Length / musicControl.pitchChangeSteps. Si lo hace, llama a IncreasePitch y tempKillCount se reinicia.

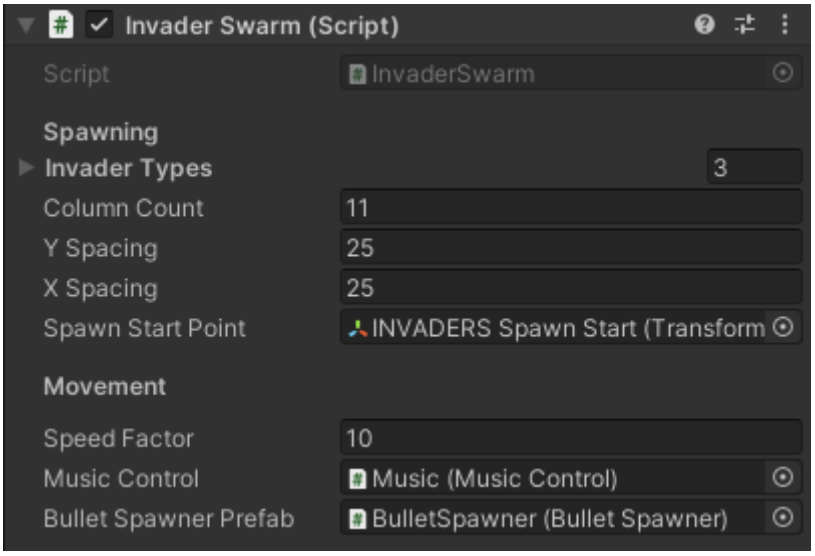
Esto significa que cuando el jugador elimina casi invaders.Length / musicControl.pitchChangeSteps invasores, tanto el tono del audio como el tempo aumentan. La variable Tempo dentro de MusicControl lleva el registro del tempo actualizado.

Finalmente, dentro de Update, reemplaza xIncrement = speedFactor * Time.deltaTime; con:

```
xIncrement = speedFactor * musicControl.Tempo * Time.deltaTime;
```

Esta línea asegura que el xIncrement considere el tempo de la música, y los invasores se muevan más rápido a medida que la música se acelera mientras el jugador elimina más y más invasores.

Guarda todo. Regresa a Unity y selecciona Game Controller. Establece el Music Control de Invader Swarm a Music. Guarda y juega.



Al disparar a los Invasores ahora vas a ver que se van acelerando.

Todavía hay un pequeño problema: Si no aciertas a ningún invasor, ellos continúan moviéndose una vez que alcanzan la parte inferior de la pantalla. Sería mejor activar "Game Over" (Fin del Juego) si la oleada alcanza la parte inferior.

Para hacer esto, abre InvaderSwarm.cs y añade lo siguiente al final de las declaraciones de variables:

```
[SerializeField]
private Transform cannonPosition;

private float minY;
private float currentY;
```

Pega este código al principio de Start:

```
currentY = spawnStartPoint.position.y;
minY = cannonPosition.position.y;
```

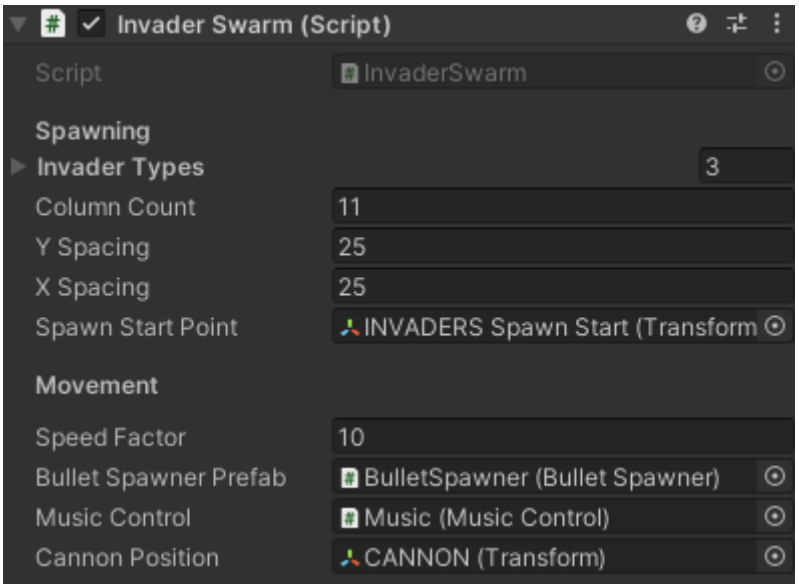
Añade estas líneas al final de ChangeDirection

```
currentY -= ySpacing;
if (currentY < minY)
{
    GameManager.Instance.TriggerGameOver();
}
```

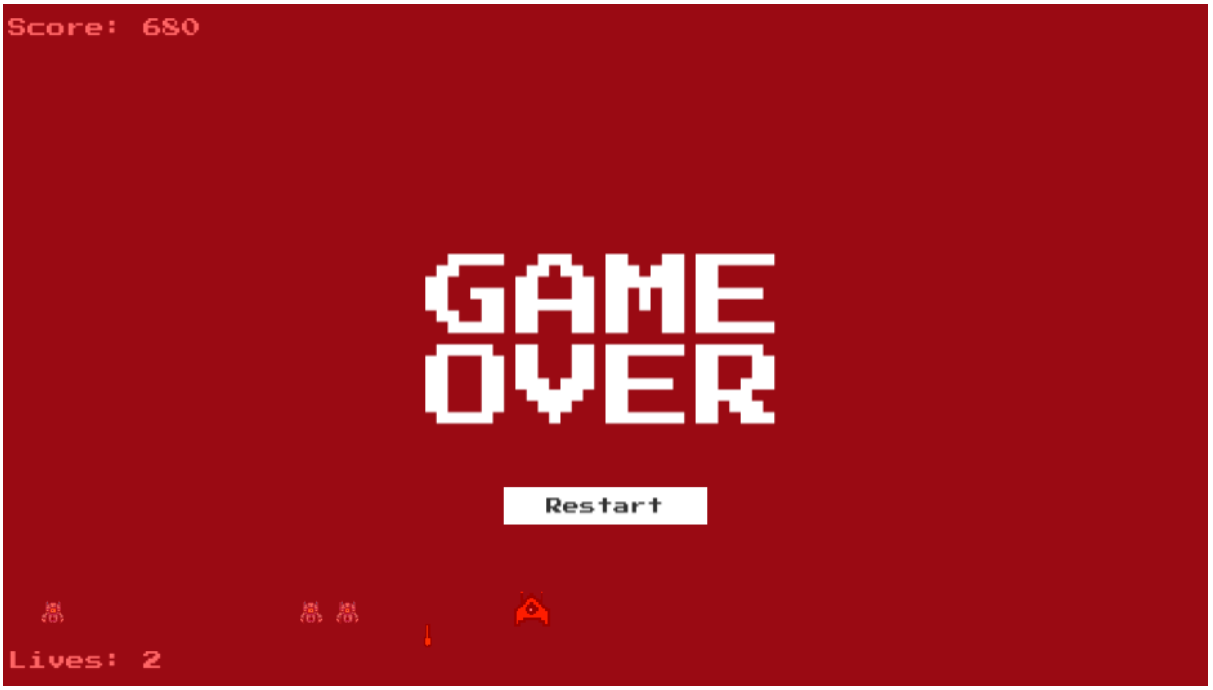
Esto es lo que hace este código:

- Dentro de Start, estableces minY a la posición Y del cañón y currentY a la posición Y de spawnStartPoint.
- Siempre que se llama, ChangeDirection decrementa currentY hasta que se vuelve menor que minY, momento en el cual el juego termina y muestra "Game Over" (Fin del Juego).

Guarda todo. Regresa a Unity y selecciona Game Controller. Establece la Cannon Position de Invader Swarm al Transform de CANNON.



Guarda y juega. Ahora, verás que el panel de "Game Over" (Fin del Juego) se activa si la oleada baja más allá de la posición del cañón.



Pues con esto llegamos al final de nuestra versión de Space Invaders

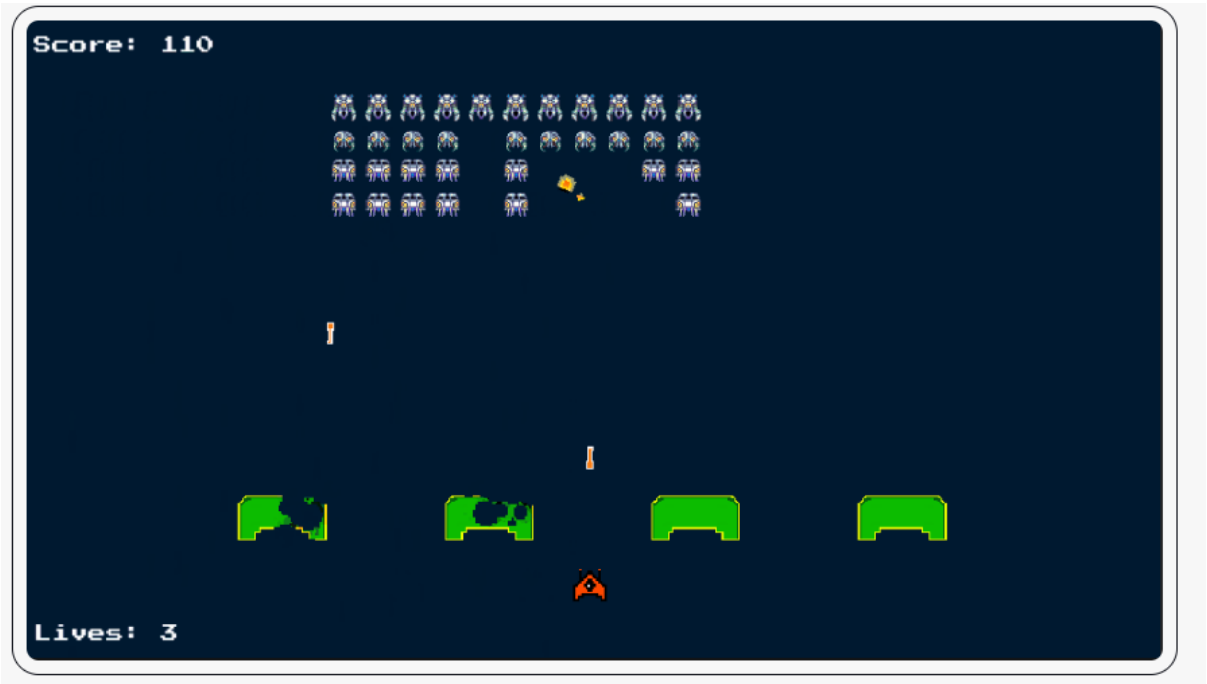
7- Mejoras del juego

A partir de este momento si te parece conveniente o quieres rematar este juego puedes realizar las siguientes mejoras que te propongo para seguir trabajando con Unity.

7.1 Añadir los Tochkas (Escudos)

Puede que hayas notado que no añadimos ningún torchka. Intenta agregarlos como un desafío.

Todo lo que necesitas ya está allí. Es posible que necesites examinar el código dentro de TorchkasManager y los scripts de Torchka. Si te quedas atascado, echa un vistazo al proyecto final para encontrar la solución. ¡Buena suerte!



7.2 Sistema de Mejoras a la Nave

Puedes crear naves espaciales que al ser destruidas den mejoras a la nave:

- Vidas extra
- Puntuación especial (x2)
- Disparo Doble

7.3 Niveles y Fases

- **Diseño de Niveles:** Crea diferentes niveles con variaciones en la formación de los invasores, patrones de movimiento, y velocidades para incrementar la dificultad y variedad en el juego.
- **Jefes de Fase:** Introduce jefes de fase al final de ciertos niveles, los cuales requieren estrategias específicas para ser derrotados. Pueden tener puntos débiles específicos y patrones de ataque únicos.