Tecnológico de Monterrey

Bullet Hell

Luis Gabriel Delfín Paulín

A01701482

03/06/2024

Descripción

Modela un jefe de un juego tipo bullet hell shooter el cual tiene 3 modos de disparo con una duración de 10 segundos por modo, es decir la duración del juego es de 30 segundos. No es necesario hacer una saturación de balas pero si busca crear patrones diferentes ya sea moviendo al jefe, moviendo los disparadores de las balas o las mismas balas.

Se debe tener un contador dentro de la pantalla de juego sobre las balas para evitar que se alente la computadora por la cantidad de pantalla.

Patrones

En esta pequeña simulación tenemos un jefe que tiene 3 patrones de disparos, donde cada patrón dura 10 segundos y cambia al siguiente patrón. Los patrones son los siguientes:

- Forma de ovni: Al ser un ovni traté de que el primer patrón simulara un cono de atracción de cuando un ovni se lleva algún objeto.
- Forma de espiral: El segundo patrón tiene forma de espiral, el cual me parece un patrón incluso más difícil de esquivar al ser menos predecible.
- Forma de círculo: El último patrón es en forma de círculo, que considero que tiene una dificultad media al ser un poco más predecible que el patrón de espiral.

Para añadir un poco más de dificultad a la simulación, nuestro ovni igual tiene un movimiento en el eje de las x. Esto para que no fuera tan predecible cada patrón.

Codificación

El programa cuenta con 6 códigos:

- BulletManager: Contador de balas para poder tener un control sobre cuántas balas hay al mismo tiempo en la interfaz del usuario.

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

0 references
public class BulletManager : MonoBehaviour
{

    // Variables para mostrar en la interfaz de usuario
    1 reference
    public TextMeshProUGUI bulletCounterText;
    3 references
    private List<GameObject> activeBullets = new List<GameObject>();

    0 references
    void Update()
    {
        // Actualizar el texto del contador
        bulletCounterText.text = "Bullets: " + activeBullets.Count;
    }

    0 references
    public void RegisterBullet(GameObject bullet)
    {
        // Registra las balas activas
        activeBullets.Add(bullet);
    }

    0 references
    public void UnregisterBullet(GameObject bullet)
    {
        // Elimina las balas de la lista
        activeBullets.Remove(bullet);
```

- EnemyBullet: Definimos el movimiento de las balas (dirección y velocidad), así como la destrucción de ellas para que no consuman almacenamiento.

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

0 references
public class EnemyBullet : MonoBehaviour
{
    1 reference
    public float speed = 5f;
    4 references
    private Vector2 direction;
    4 references
    private BulletManager bulletManager;

    0 references
    void Start()
    {
        if (direction == Vector2.zero)
        {
            direction = Vector2.down;
        }

        // Contador de balas
        bulletManager = FindObjectOfType<BulletManager>();
        bulletManager.RegisterBullet(gameObject);
    }

    0 references
    void Update()
    {
        Move();
    }
```

- EnemyMovement: Movemos al ovni de izquierda a derecha, para agregar un poco de dificultad al jefe.

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

0 references
public class EnemyMovement : MonoBehaviour
{
    2 references
    public float sideToSideSpeed = 2f;
    2 references
    public float sideToSideDistance = 5f;

    3 references
    private bool isMovingRight = true;
    3 references
    private Vector3 initialPosition;

    0 references
    void Start()
    {
        initialPosition = transform.position;
    }

    // Movemos en el eje x
    0 references
    void Update()
    {
        if (isMovingRight)
        {
            transform.Translate(Vector3.right * sideToSideSpeed * Time.deltaTime);
            if (transform.position.x >= initialPosition.x + sideToSideDistance)
```

- EnemyShooter: Asignamos a nuestro ovni para que tenga diferentes patrones de disparos que se van alternando de manera regular.

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

0 references
public class EnemyShooter : MonoBehaviour
{
    3 references
    public GameObject bulletPrefab;
    1 reference
    public float fireRate = 0.5f;
    2 references
    private float nextFireTime = 0f;
    3 references
    private int currentPattern = 0;
    2 references
    private float patternChangeInterval = 10f;
    3 references
    private float nextPatternChangeTime = 0f;
    5 references
    private float spiralAngle = 0f;

    // Referencia al componente de movimiento de la nave enemiga
    1 reference
    private EnemyMovement enemyMovement;

    0 references
    void Start()
    {
        nextPatternChangeTime = Time.time + patternChangeInterval;

        // Obtener la referencia al componente de movimiento de la nave enemiga
        enemyMovement = GetComponent<EnemyMovement>();
```

- PlayerController: Tenemos aquí él código básico de movimiento del jugador y además limitamos el movimiento a la posición de la cámara para evitar que lo perdamos de vista.

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

0 references
public class PlayerController : MonoBehaviour
{
    1 reference
    public float moveSpeed = 1f;

    // Update is called once per frame
    0 references
    void Update()
    {
        Move();
    }

    1 reference
    void Move()
    {
        float moveX = Input.GetAxis("Horizontal");
        float moveY = Input.GetAxis("Vertical");

        Vector3 moveDirection = new Vector3(moveX, moveY, 0);
        transform.position += moveDirection * moveSpeed * Time.deltaTime;

        // Limitar la posición del jugador dentro de los límites de la cámara
        float halfPlayerWidth = transform.localScale.x / 2f;
        float halfPlayerHeight = transform.localScale.y / 2f;
        float cameraDistance = transform.position.z - Camera.main.transform.position.z;
```

- PlayerHealth: Código para destruir nuestro GameObject de jugador.

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

0 references
public class PlayerHealth : MonoBehaviour
{
    0 references
    public void DestroyShip()
    {
        Destroy(gameObject);
    }
}
```

Vídeo

https://youtu.be/uRq2DusE8n0

Github

https://github.com/Gabriel-Delfin/BulletHell

Reflexión

Esta actividad me ayudó bastante a tener noción de cómo se crea un pequeño juego, incluyendo movimientos de balas creados con prefabs y destruirlos cuando ya no se necesitan, con el fin de no saturar la memoria. Creé igual un jefe que se movía junto con las balas y si las balas llegaban a tocar el jugador, el GameObject se destruye. Igual fue un reto crear los patrones de las balas, donde se utilizan matemáticas.