Sistema de Generación de Enemigos

Ciro Sebastián Hernández López

Tabla de contenidos

- **Que es**
- **Como funciona**
- **Como se usa**

Que es

Esta es una herramienta con el objetivo de generar enemigos aleatorios en la escena del juego y mostrar información sobre los mismos en una serie de Textos en la interfaz de usuario.

Como Funciona

Primero, se declaran variables públicas para los prefabs de los enemigos, la cantidad máxima de enemigos, los tiempos de espera mínimo y máximo, y la suerte mínima y máxima para la generación de enemigos. También hay variables públicas para los Textos que se mostrarán en la interfaz de usuario.

```
public GameObject EnemigoPrefab;
public GameObject EnemigoGrandePrefab;

public float enemyNum;
public int maxNum = 10;

public float tiempoMinimo = 1f;
public float tiempoMaximo = 3f;

public int minSuerte = 1;
public int maxSuerte = 5;
```

```
public Text prefab;
public Text wait;
public Text enemyNumber;
public Text enemyMax;
public Text enemyPos;
public Text luck;
public Text luckRange;
public Text waitMin;
public Text waitMax;
```

Luego, en la función Start(), se llama a la función IEnumerator Spawn() usando StartCoroutine(). También se inicializan los Textos de la interfaz de usuario a "Nulo" y se muestran algunos mensajes informativos.

```
void Start()
{

    StartCoroutine(Spawn());
    prefab.text = "Nulo";
    wait.text = "Nulo";
    enemyPos.text = "Nulo";
    luck.text = "Nulo";
    luckRange.text = "Rango de suerte: " + minSuerte.ToString() + " a " + maxSuerte.ToString();
    waitMin.text = "Espera minima:" + tiempoMinimo.ToString();
    waitMax.text = "Espera maxima:" + tiempoMaximo.ToString();
}
```

La función IEnumerator Spawn() es la que se encarga de generar los enemigos de forma aleatoria. Primero, se establece un tiempo de espera aleatorio entre el tiempo mínimo y máximo declarados al principio del script. Luego, se espera ese tiempo usando la función WaitForSeconds().

```
3 referencias
private IEnumerator Spawn()
{
    float espera = Random.Range(tiempoMinimo, tiempoMaximo);
    yield return new WaitForSeconds(espera);
    wait.text = "Se esperaron: " + espera.ToString() + " segundos";
```

Después, se genera un número aleatorio de suerte entre el mínimo y máximo declarados al principio del script.

```
int suerte = Random.Range(minSuerte,maxSuerte);
```

Si la cantidad actual de enemigos en la escena es menor que la cantidad máxima permitida, entonces se genera un enemigo.

La probabilidad de generar un enemigo pequeño o grande depende del número de suerte generado. Si el número de suerte es menor que 4, entonces se genera un enemigo pequeño en una posición aleatoria en el plano x-z con una altura fija de 1 unidad. De lo contrario, se genera un enemigo grande en una posición aleatoria en el plano x-z con una altura fija de 4 unidades.

Después de generar el enemigo, se llama a la función Spawn() de nuevo usando StartCoroutine() para que el ciclo de generación continúe. También se actualizan los Textos de la interfaz de usuario con información sobre el tipo de enemigo generado, la posición del enemigo y la cantidad de suerte.

```
if(enemyNum < maxNum)
{
    if(suerte < 4)
    {
        Vector3 posicion = new Vector3(Random.Range(-5f, 5f), 1, Random.Range(-6f, 6f));
        GameObject newEnemy = Instantiate(EnemigoPrefab, posicion, Quaternion.identity);

        StartCoroutine(Spawn());

        prefab.text = "Tipo de Enemigo: Pequeño";
        enemyPos.text = "Posicion: " + posicion.ToString();
    }
    else
    {
        Vector3 posicion = new Vector3(Random.Range(-5f, 5f), 4, Random.Range(-6f, 6f));
        GameObject newEnemy = Instantiate(EnemigoGrandePrefab, posicion, Quaternion.identity);

        StartCoroutine(Spawn());

        prefab.text = "Tipo de Enemigo: Grande";
        enemyPos.text = "Posicion: " + posicion.ToString();
    }
}
enemyNum++;

luck.text = "Suerte: " + suerte.ToString();
```

Finalmente, en la función Update(), se actualizan los Textos de la interfaz de usuario con información sobre la cantidad de enemigos actual y el máximo permitido.

```
private void Update()
{
    enemyNumber.text = "Numero de enemigos: " + enemyNum.ToString();
    enemyMax.text = "Mximo de enemigos:" + maxNum.ToString();
}
```

Los enemigos creados tienen un script sencillo que hace que sigan al jugador

```
public class Enemigo : MonoBehaviour
{
    public float speed = 3.0f;
    private Transform player;


    Mensaje de Unity | 0 referencias
    void Start()
    {
        player = GameObject.FindWithTag("Player").transform;
    }

    Mensaje de Unity | 0 referencias
    void Update()
    {
        transform.LookAt(player);
        transform.Translate(Vector3.forward * speed * Time.deltaTime);
    }
}
```

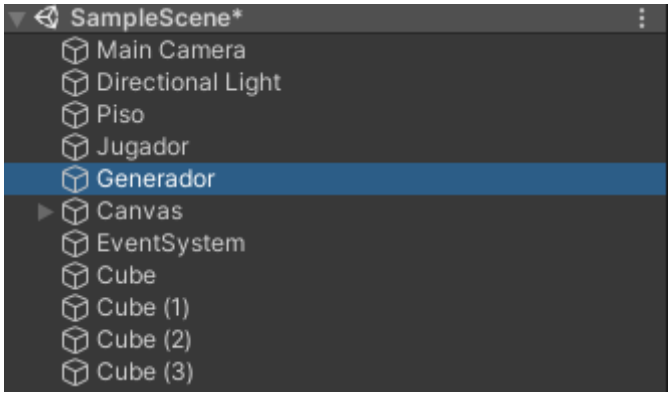
En el método "Start()", se inicializa la variable "player" buscando en el juego el objeto con la etiqueta "Player".

En el método "Update()", se ejecutan dos acciones en cada frame. Primero, la función "LookAt()" hace que el enemigo siempre mire hacia el objeto del jugador. Luego, se mueve hacia adelante en la dirección a la que está mirando utilizando la función "Translate()" y la dirección "Vector3.forward", que representa el eje Z en el espacio tridimensional. La velocidad del enemigo se controla mediante la variable "speed" y el tiempo transcurrido desde el último fotograma se toma en cuenta con "Time.deltaTime".

En general, este script hace que el enemigo siga al jugador y se mueva hacia él a una velocidad determinada.

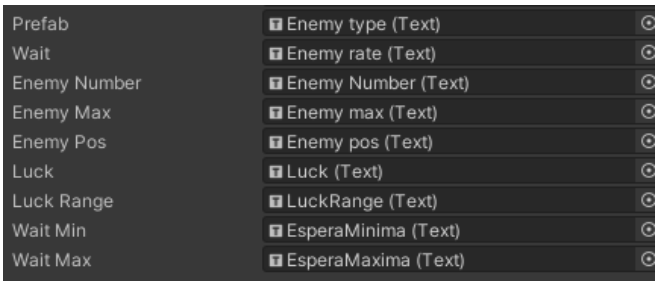
Como Usarlo

Para iniciar una debe crear un objeto en la escena unity, en este caso un objeto vacío.



Se le debe agregar el script de "Genrador" (Lamento la falta de ortografía en el nombre del script), después se deben agregar los prefabs deseados para que este los genere.

Para que muestre la información en la interfaz se deben crear una serie de textos en un canvas y agregarlos al inspector.



Una vez hecho esto se empezarán a generar automáticamente los enemigos en lapsos de tiempo y posiciones aleatorias.

Se pueden modificar casi todos los aspectos de la generación desde el inspector de unity.