using System.Collections;

using System.Collections.Generic;

using TMPro;

using Unity.VisualScripting;

using UnityEngine;

public class SpawnEnemies : MonoBehaviour

{

[System.Serializable]

public class EnemySpawnInfo

{

public int aventureros;

public int tramperos;

public bool tieneHeroe;

}

public GameObject canvasSpawnPointPrefab;

public GameObject canvasHeroeInfoPrefab;

public GameObject enemyPrefab, heroePrefab, tramperoPrefab;

public GameObject[] spawnWaypoints;

private List<GameObject> spawnListPoints = new List<GameObject>();

private List<GameObject> chosenPoints = new List<GameObject>();

private Dictionary<GameObject, EnemySpawnInfo> enemigosPorPuerta;

private Transform heroePoint;

bool isSpecialRound = false;

// Start is called once before the first execution of Update after the MonoBehaviour is created

void Start()

{

AddWayPoints(spawnWaypoints, null);

}

// Update is called once per frame

void Update()

{

}

public void GenerarPuntosSpawn(int roundNumber, int enemiesToKillInCurrentWave)

{

chosenPoints.Clear();

//n mero de puntos de spawn

int numberOfSpawnPoints = roundNumber >= 20 ? 4 : Mathf.Clamp((roundNumber - 1) / 5 + 1, 1, 4);

// Crear una copia temporal de la lista original

List<GameObject> tempList = new List<GameObject>(spawnListPoints);

// Algoritmo Fisher-Yates para mezclar la lista

for (int i = 0; i < tempList.Count; i++)

{

int randomIndex = Random.Range(i, tempList.Count);

GameObject temp = tempList[randomIndex];

tempList[randomIndex] = tempList[i];

tempList[i] = temp;

}

// Tomar los primeros 'numberOfSpawnPoints' elementos

for (int i = 0; i < numberOfSpawnPoints && i < tempList.Count; i++)

{

chosenPoints.Add(tempList[i]);

}

ActivarLuces();

CalcularNumeroEnemigos(roundNumber, enemiesToKillInCurrentWave);

}

private void ActivarLuces()

{

foreach (GameObject spawnPoint in chosenPoints)

{

for (var i = 0; i < spawnPoint.transform.childCount; i++)

{

var torch = spawnPoint.transform.GetChild(i);

// activar la animaci n de la antorcha

if(torch.GetComponent<Animator>() != null) torch.GetComponent<Animator>().SetBool("enabled", true);

//activar su luz

torch.GetChild(0).gameObject.SetActive(true);

}

}

}

public void DesactivarLuces()

{

foreach (GameObject spawnPoint in chosenPoints)

{

for (var i = 0; i < spawnPoint.transform.childCount; i++)

{

GameObject go = spawnPoint.transform.GetChild(i).gameObject;

if (go.GetComponent<Animator>() != null)

{

go.GetComponent<Animator>().SetBool("enabled", false);

}

go.transform.GetChild(0).gameObject.SetActive(false);

}

}

}

void CalcularNumeroEnemigos(int roundNumber, int enemiesToKillInCurrentWave)

{

isSpecialRound = (roundNumber % 5 == 0);

enemigosPorPuerta = new Dictionary<GameObject, EnemySpawnInfo>();

// Inicializar diccionario con informaci n vac a para cada puerta

foreach (GameObject puerta in chosenPoints)

{

enemigosPorPuerta[puerta] = new EnemySpawnInfo();

}

// Repartir enemigos entre las puertas

for (int i = 1; i <= enemiesToKillInCurrentWave; i++)

{

if (chosenPoints.Count == 0)

continue;

int randomWP = Random.Range(0, chosenPoints.Count);

GameObject chosenPoint = chosenPoints[randomWP];

// Determinar si ser aventurero o trampero

// Probabilidad: 70% aventurero, 30% trampero

float random = Random.Range(0f, 1f);

if (random < 0.7f)

{

enemigosPorPuerta[chosenPoint].aventureros++;

}

else

{

enemigosPorPuerta[chosenPoint].tramperos++;

}

}

// Verificar que cada puerta tenga m s aventureros que tramperos

foreach (var puerta in enemigosPorPuerta.Keys)

{

EnemySpawnInfo info = enemigosPorPuerta[puerta];

// Si hay tramperos pero no hay aventureros, convertir un trampero en aventurero

if (info.tramperos > 0 && info.aventureros == 0)

{

info.aventureros = 1;

info.tramperos--;

}

// Si hay m s tramperos que aventureros, equilibrar

else if (info.tramperos >= info.aventureros && info.aventureros > 0)

{

int exceso = info.tramperos - info.aventureros + 1;

info.aventureros += exceso;

info.tramperos -= exceso;

// Asegurar que no queden tramperos negativos

if (info.tramperos < 0)

{

info.aventureros += info.tramperos;

info.tramperos = 0;

}

}

}

// Manejar h roe en rondas especiales

if (isSpecialRound)

{

FindFirstObjectByType<GameManager>().enemiesToKillInCurrentWave++;

int randomWP = Random.Range(0, chosenPoints.Count);

heroePoint = chosenPoints[randomWP].transform;

enemigosPorPuerta[heroePoint.gameObject].tieneHeroe = true;

}

ActivarPanelInfo();

}

void ActivarPanelInfo()

{

foreach (var puerta in chosenPoints)

{

GameObject canvas = Instantiate(canvasSpawnPointPrefab, puerta.transform);

EnemySpawnInfo info = enemigosPorPuerta[puerta];

// Panel principal (Aventureros)

TextMeshProUGUI textoPanel = canvas.GetComponentInChildren<TextMeshProUGUI>();

textoPanel.text = "x" + info.aventureros.ToString();

// Panel del H roe

Transform panelHeroe = canvas.transform.GetChild(0).Find("PanelInfoHeroe");

if (info.tieneHeroe)

{

panelHeroe.gameObject.SetActive(true);

// Asumiendo que el panel del h roe tambi n tiene un texto para mostrar "x1"

TextMeshProUGUI textoHeroe = panelHeroe.GetComponentInChildren<TextMeshProUGUI>();

if (textoHeroe != null)

textoHeroe.text = "x1";

}

else

{

panelHeroe.gameObject.SetActive(false);

}

// Panel del Trampero

Transform panelTrampero = canvas.transform.GetChild(0).Find("PanelInfoTrampero");

if (info.tramperos > 0)

{

panelTrampero.gameObject.SetActive(true);

TextMeshProUGUI textoTrampero = panelTrampero.GetComponentInChildren<TextMeshProUGUI>();

if (textoTrampero != null)

textoTrampero.text = "x" + info.tramperos.ToString();

}

else

{

panelTrampero.gameObject.SetActive(false);

}

}

}

public IEnumerator GenerarEnemigos()

{

// ---------------- INSTANCIAR EL H ROE CADA 5 RONDAS -------------------

if (isSpecialRound)

{

//Debug.Log("Spawning HEROE");

Instantiate(heroePrefab, heroePoint.position, Quaternion.identity);

yield return new WaitForSeconds(Random.Range(0.5f, 1f));

}

foreach (var puerta in enemigosPorPuerta.Keys)

{

EnemySpawnInfo info = enemigosPorPuerta[puerta];

//Debug.Log($"Puerta: Aventureros={info.aventureros}, Tramperos={info.tramperos}");

// Crear una lista mezclada de tipos de enemigos para esta puerta

List<string> tiposEnemigos = new List<string>();

// A adir aventureros

for (int i = 0; i < info.aventureros; i++)

{

tiposEnemigos.Add("aventurero");

}

// A adir tramperos

for (int i = 0; i < info.tramperos; i++)

{

tiposEnemigos.Add("trampero");

}

//Debug.Log($"Lista tipos antes de mezclar: {string.Join(", ", tiposEnemigos)}");

// Mezclar la lista para orden aleatorio

for (int i = 0; i < tiposEnemigos.Count; i++)

{

int randomIndex = Random.Range(i, tiposEnemigos.Count);

string temp = tiposEnemigos[randomIndex];

tiposEnemigos[randomIndex] = tiposEnemigos[i];

tiposEnemigos[i] = temp;

}

//Debug.Log($"Lista tipos despu s de mezclar: {string.Join(", ", tiposEnemigos)}");

// Instanciar enemigos en orden mezclado

foreach (string tipoEnemigo in tiposEnemigos)

{

yield return new WaitForSeconds(Random.Range(0.5f, 1f));

Vector2 offset = new Vector2(Random.Range(-1.25f, 1.25f), Random.Range(-1.25f, 1.25f));

Vector2 spawnPosition = new Vector2(puerta.transform.position.x, puerta.transform.position.y) + offset;

//Debug.Log($"Spawning {tipoEnemigo} at {spawnPosition}");

if (tipoEnemigo == "aventurero")

{

//Debug.Log($"Instanciando AVENTURERO con prefab: {enemyPrefab.name}");

GameObject spawnedEnemy = Instantiate(enemyPrefab, spawnPosition, Quaternion.identity);

// VERIFICAR EL TIPO EN EL ENEMYCONTROLLER

//EnemyController controller = spawnedEnemy.GetComponent<EnemyController>();

//if (controller != null)

//{

// Debug.Log($"EnemyController tipo: {controller.type}");

//}

}

else if (tipoEnemigo == "trampero")

{

//Debug.Log($"Instanciando TRAMPERO con prefab: {tramperoPrefab.name}");

GameObject spawnedEnemy = Instantiate(tramperoPrefab, spawnPosition, Quaternion.identity);

// VERIFICAR EL TIPO EN EL ENEMYCONTROLLER

//EnemyController controller = spawnedEnemy.GetComponent<EnemyController>();

//if (controller != null)

//{

// Debug.Log($"EnemyController tipo: {controller.type}");

//}

}

}

}

}

public void AddWayPoints(GameObject[] posiciones, GameObject[] removePoints)

{

if (posiciones == null && removePoints == null) return;

RemoveWayPoints(removePoints);

foreach (GameObject point in posiciones)

if (!spawnListPoints.Contains(point))

spawnListPoints.Add(point);

}

void RemoveWayPoints(GameObject[] removePoints)

{

if (removePoints == null)

return;

foreach (GameObject point in removePoints)

{

spawnListPoints.Remove(point);

point.SetActive(false);

}

}

}