1. Scripts
2. Player
3. Enemy
4. Cấu trúc lớp Enemy

1. Mục đích

Đây là script quản lý hành vi và trạng thái của kẻ địch (Enemy) trong game Unity, bao gồm máu, khả năng bắn, hiệu ứng khi bị trúng đạn và khi bị phá hủy.

2. Các biến (FIELDS)

• health: Số máu của Enemy.

• Projectile: Prefab viên đạn mà Enemy sẽ bắn ra.

• destructionVFX: Hiệu ứng khi Enemy bị phá hủy.

• hitEffect: Hiệu ứng khi Enemy bị trúng đạn.

• shotChance: Tỉ lệ phần trăm để Enemy bắn đạn khi di chuyển.

• shotTimeMin, shotTimeMax: Thời gian ngẫu nhiên để Enemy bắt đầu bắn.

3. Hàm Start()

• Khi Enemy xuất hiện, sẽ gọi hàm ActivateShooting sau một khoảng thời gian ngẫu nhiên từ shotTimeMin đến shotTimeMax.

4. Hàm ActivateShooting()

• Kiểm tra xác suất bắn (shotChance). Nếu đạt, Enemy sẽ tạo ra một viên đạn tại vị trí hiện tại.

5. Hàm GetDamage(int damage)

• Enemy bị trừ máu khi nhận sát thương.

• Nếu máu ≤ 0, gọi hàm Destruction() để phá hủy Enemy.

• Nếu còn máu, tạo hiệu ứng trúng đạn (hitEffect).

6. Hàm OnTriggerEnter2D(Collider2D collision)

• Khi Enemy va chạm với Player, Player sẽ nhận sát thương bằng sát thương của viên đạn Enemy (nếu có), nếu không thì nhận sát thương mặc định là 1.

1. Cấu trúc mã nguồn

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

/// <summary>

/// Script này định nghĩa máu (health) và hành vi của kẻ địch ('Enemy').

/// </summary>

public class Enemy : MonoBehaviour {

#region FIELDS

[Tooltip("Health points in integer")] // Điểm máu (health) dưới dạng số nguyên

public int health;

[Tooltip("Enemy's projectile prefab")] // Prefab đạn của kẻ địch

public GameObject Projectile;

[Tooltip("VFX prefab generating after destruction")] // Prefab hiệu ứng VFX sau khi bị tiêu diệt

public GameObject destructionVFX;

public GameObject hitEffect;

[HideInInspector] public int shotChance; //Tỉ lệ xác suất kẻ địch bắn khi di chuyển trên đường đi

[HideInInspector] public float shotTimeMin, shotTimeMax; // Thời gian tối thiểu và tối đa để bắt đầu bắn từ lúc bắt đầu đường đi

#endregion

private void Start()

{

Invoke("ActivateShooting", Random.Range(shotTimeMin, shotTimeMax));

}

// Coroutine thực hiện hành động bắn

void ActivateShooting()

{

if (Random.value < (float)shotChance / 100)) //Nếu giá trị ngẫu nhiên nhỏ hơn xác suất bắn, thì sẽ bắn {

Instantiate(Projectile, gameObject.transform.position, Quaternion.identity);

}

}

// Phương thức nhận sát thương cho kẻ địch

public void GetDamage(int damage)

{

health -= damage; //Giảm máu theo giá trị sát thương, Nếu máu nhỏ hơn hoạc bằng 0 thì bắt đầu quy trình tiêu diệt

if (health <= 0)

Destruction();

else

Instantiate(hitEffect,transform.position,Quaternion.identity,transform);

}

//Nếu kẻ địch va chạm với người chơi, người chơi sẽ nhận sát thương bằng với sát thương của đạn

private void OnTriggerEnter2D(Collider2D collision)

{

if (collision.tag == "Player")

{

if (Projectile.GetComponent<Projectile>() != null)

Player.instance.GetDamage(Projectile.GetComponent<Projectile>().damage);

else

Player.instance.GetDamage(1);

}

}

//Phương thức tiêu diệt kể địch.

void Destruction()

{

Instantiate(destructionVFX, transform.position, Quaternion.identity);

Destroy(gameObject);

}

}

1. Player Moving
2. Player Shooting
3. Bonus
4. Boudary
5. Direct Moving
6. Follow The Path
7. Phân tích cấu trức
8. Khai báo biến

• path: Mảng các điểm (Transform) tạo thành đường đi cho Enemy.

• speed: Tốc độ di chuyển của Enemy trên đường đi.

• rotationByPath: Nếu true, Enemy sẽ xoay theo hướng di chuyển trên đường.

• loop: Nếu true, Enemy sẽ quay lại điểm bắt đầu khi hoàn thành đường đi, nếu false thì Enemy sẽ bị xóa khi kết thúc đường.

• currentPathPercent: Phần trăm tiến trình di chuyển trên đường (0 là bắt đầu, 1 là kết thúc).

• pathPositions: Mảng các điểm đường đi dưới dạng Vector3.

• movingIsActive: Kiểm soát Enemy có đang di chuyển hay không.

---

1. Hàm SetPath()

• Khởi tạo lại tiến trình di chuyển.

• Chuyển các điểm đường đi từ Transform sang Vector3.

• Đặt Enemy về vị trí bắt đầu của đường đi.

• Đặt lại hướng xoay nếu không xoay theo đường.

• Bật trạng thái di chuyển.

---

1. Hàm Update()

• Nếu Enemy đang di chuyển, cập nhật tiến trình di chuyển dựa trên tốc độ và thời gian.

• Cập nhật vị trí Enemy theo đường đi.

• Nếu bật xoay, Enemy sẽ xoay theo hướng di chuyển.

• Khi hoàn thành đường đi: nếu loop thì quay lại đầu, nếu không thì xóa Enemy.

---

1. Hàm NewPositionByPath()

• Tính toán vị trí mới của Enemy trên đường đi dựa vào phần trăm tiến trình.

---

1. Hàm Interpolate()

• Tính toán vị trí trên đường đi bằng phương pháp Catmull-Rom spline (nội suy mượt giữa các điểm).

---

1. Hàm CreatePoints()

• Tạo thêm điểm đầu và cuối để spline hoạt động mượt mà.

• Nếu đường đi là vòng lặp, điều chỉnh lại điểm đầu/cuối cho phù hợp.

---

Tóm lại:

Script này giúp Enemy di chuyển mượt mà theo đường đi định sẵn, có thể xoay theo hướng di chuyển và lặp lại đường đi nếu cần. Các hàm nội suy và tạo điểm giúp chuyển động không bị giật, tạo cảm giác tự nhiên.

1. Cấu trúc mã nguồn

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using System;

/// <summary>

/// Script này điều khiển kẻ địch (‘Enemy’) di chuyển theo một đường dẫn được xác định.

/// </summary>

public class FollowThePath : MonoBehaviour

{

[HideInInspector] public Transform[] path; //Các điểm trên đường đi mà kẻ địch sẽ đi qua

[HideInInspector] public float speed;

[HideInInspector] public bool rotationByPath; //Kẻ địch có xoay theo hướng của đường đi hay không

[HideInInspector] public bool loop; //Nếu loop là true, kẻ địch sẽ quay lại điểm bắt đầu sau khi hoàn thành đường đi

float currentPathPercent; //Phần trăm hoàn thành đường đi hiện tại

Vector3[] pathPositions; //Các điểm trên đường đi dưới dạng Vector3

[HideInInspector] public bool movingIsActive; //Kẻ địch có đang di chuyển hay không

//Thiết lập các thông số đường đi cho kẻ địch và đưa kẻ địch đến điểm bắt đầu của đường đi

public void SetPath()

{

currentPathPercent = 0;

pathPositions = new Vector3[path.Length]; //Chuyển các điểm trên đường đi sang Vector3

for (int i = 0; i < pathPositions.Length; i++)

{

pathPositions[i] = path[i].position;

}

transform.position = NewPositionByPath(pathPositions, 0); //Đưa kẻ địch đến điểm bắt đầu của đường đi

if (!rotationByPath)

transform.rotation = Quaternion.identity;

movingIsActive = true;

}

private void Update()

{

if (movingIsActive)

{

currentPathPercent += speed / 100 \* Time.deltaTime; //Mỗi lần cập nhật, tính phần trăm đường đi đã hoàn thành dựa theo tốc độ

transform.position = NewPositionByPath(pathPositions, currentPathPercent); //Di chuyển kẻ địch đến vị trí trên đường đi, tính bằng phương thức NewPositionByPath

if (rotationByPath) //Xoay kẻ địch theo hướng của đường đi, nếu đã bật rotationByPath

{

transform.right = Interpolate(CreatePoints(pathPositions), currentPathPercent + 0.01f) - transform.position;

transform.Rotate(Vector3.forward \* 90);

}

if (currentPathPercent > 1) //Khi hoàn thành đường đi

{

if (loop) //Nếu bật loop, quay lại điểm bắt đầu; nếu không, tiêu diệt hoặc vô hiệu hoá kẻ địch

currentPathPercent = 0;

else

{

Destroy(gameObject);

}

}

}

}

Vector3 NewPositionByPath(Vector3[] pathPos, float percent)

{

return Interpolate(CreatePoints(pathPos), currentPathPercent);

}

Vector3 Interpolate(Vector3[] path, float t)

{

int numSections = path.Length - 3;

int currPt = Mathf.Min(Mathf.FloorToInt(t \* numSections), numSections - 1);

float u = t \* numSections - currPt;

Vector3 a = path[currPt];

Vector3 b = path[currPt + 1];

Vector3 c = path[currPt + 2];

Vector3 d = path[currPt + 3];

return 0.5f \* ((-a + 3f \* b - 3f \* c + d) \* (u \* u \* u) + (2f \* a - 5f \* b + 4f \* c - d) \* (u \* u) + (-a + c) \* u + 2f \* b);

}

Vector3[] CreatePoints(Vector3[] path)

{

Vector3[] pathPositions;

Vector3[] newPathPos;

int dist = 2;

pathPositions = path;

newPathPos = new Vector3[pathPositions.Length + dist];

Array.Copy(pathPositions, 0, newPathPos, 1, pathPositions.Length);

newPathPos[0] = newPathPos[1] + (newPathPos[1] - newPathPos[2]);

newPathPos[newPathPos.Length - 1] = newPathPos[newPathPos.Length - 2] + (newPathPos[newPathPos.Length - 2] - newPathPos[newPathPos.Length - 3]);

if (newPathPos[1] == newPathPos[newPathPos.Length - 2])

{

Vector3[] LoopSpline = new Vector3[newPathPos.Length];

Array.Copy(newPathPos, LoopSpline, newPathPos.Length);

LoopSpline[0] = LoopSpline[LoopSpline.Length - 3];

LoopSpline[LoopSpline.Length - 1] = LoopSpline[2];

newPathPos = new Vector3[LoopSpline.Length];

Array.Copy(LoopSpline, newPathPos, LoopSpline.Length);

}

return (newPathPos);

}

}

1. Leve Controller
2. Pooling Contronller
3. Projectile
4. Repeating Background
5. Visual Effects
6. Wave
7. Prefabs
8. Scenes
9. Background
10. Planets
11. VFX
12. Projectiles
13. Animation
14. TextMesh