**Enemy script**

**MeleeEnemy.cs**

using UnityEngine;

public class MeleeEnemy : MonoBehaviour

{

[SerializeField] private CapsuleCollider2D capsuleCollider;

[SerializeField] private LayerMask playerLayer;

[SerializeField] private Enemy\_group MeleeEnemy\_details;

private float attackCooldown;

private float range;

private float distance;

private int damage;

private int criticalDamage;

private float cooldownTimer = Mathf.Infinity;

private int randomAttack;

private Animator anim;

private Health playerHealth;

private Patroll enemyPatrol;

private void Awake()

{

anim = GetComponent<Animator>();

enemyPatrol = GetComponentInParent<Patroll>();

}

private void Start()

{

attackCooldown = MeleeEnemy\_details.attackCooldown;

range = MeleeEnemy\_details.detectRange;

distance = MeleeEnemy\_details.detectDistance;

damage = MeleeEnemy\_details.damage;

criticalDamage = MeleeEnemy\_details.criticalDamage;

}

private void Update()

{

cooldownTimer += Time.deltaTime;

//attack only player in sight

if (PlayerInSight())

{

if (cooldownTimer >= attackCooldown)

{

//attack

if (!playerHealth.dead)

{

cooldownTimer = 0;

if (randomAttack <= 3) anim.SetTrigger("Attack1");

else anim.SetTrigger("Attack2");

}

}

}

if(enemyPatrol != null) enemyPatrol.enabled = !PlayerInSight();

}

private bool PlayerInSight()

{

RaycastHit2D hit = Physics2D.BoxCast(capsuleCollider.bounds.center + transform.right \* distance \* (transform.localScale.x / Mathf.Abs(transform.localScale.x)),

new Vector3(capsuleCollider.bounds.size.x \* range, capsuleCollider.bounds.size.y, capsuleCollider.bounds.size.z), 0, Vector2.left, 0, playerLayer);

if (hit.collider != null) playerHealth = hit.collider.GetComponent<Health>();

return (hit.collider != null);

}

private void DamagePlayer()

{

if (PlayerInSight())

{

//Damage player health

randomAttack = Random.Range(0, 10);

if (randomAttack <= 3) playerHealth.TakeDamage(damage);

else playerHealth.TakeDamage(criticalDamage);

}

}

private void OnDrawGizmos()

{

Gizmos.color = Color.red;

Gizmos.DrawWireCube(capsuleCollider.bounds.center + transform.right \* distance \* (transform.localScale.x / Mathf.Abs(transform.localScale.x)),

new Vector3(capsuleCollider.bounds.size.x \* range, capsuleCollider.bounds.size.y, capsuleCollider.bounds.size.z));

}

}

Đoạn mã này xử lý hành vi của một enemy cận chiến trong trò chơi Unity. Enemy này có thể phát hiện người chơi trong phạm vi xác định, tấn công người chơi khi có thể và tuần tra khi không phát hiện thấy người chơi.

Biến thành viên:

* CapsuleCollider2D capsuleCollider: Bộ va chạm của enemy, được sử dụng để phát hiện người chơi.
* LayerMask playerLayer: Lớp đối tượng của người chơi, dùng để kiểm tra va chạm.
* Enemy\_group MeleeEnemy\_details: Thông tin chi tiết về enemy cận chiến, bao gồm thời gian hồi chiêu, phạm vi phát hiện, khoảng cách phát hiện, sát thương và sát thương chí mạng.
* CapsuleCollider2D capsuleCollider: Bộ va chạm của enemy, được sử dụng để phát hiện người chơi.
* LayerMask playerLayer: Lớp đối tượng của người chơi, dùng để kiểm tra va chạm.
* Enemy\_group MeleeEnemy\_details: Thông tin chi tiết về enemy cận chiến, bao gồm thời gian hồi chiêu, phạm vi phát hiện, khoảng cách phát hiện, sát thương và sát thương chí mạng.
* float attackCooldown: Thời gian hồi chiêu giữa các đòn tấn công.
* float range: Phạm vi phát hiện người chơi.
* float distance: Khoảng cách phát hiện người chơi.
* int damage: Sát thương của đòn tấn công thông thường.
* int criticalDamage: Sát thương của đòn tấn công chí mạng.
* float cooldownTimer: Bộ đếm thời gian để theo dõi thời gian hồi chiêu, khởi đầu bằng giá trị vô cực.
* int randomAttack: Biến ngẫu nhiên để xác định loại đòn tấn công.
* Animator anim: Thành phần Animator dùng để điều khiển hoạt ảnh của enemy.
* Health playerHealth: Thành phần Health của người chơi, được sử dụng để giảm máu của người chơi khi bị tấn công.
* Patroll enemyPatrol: Thành phần Patroll dùng để điều khiển hành vi tuần tra của enemy.

**Awake():** Phương thức này được gọi khi đối tượng được khởi tạo. Nó lấy các thành phần Animator và Patroll từ đối tượng enemy.

**Start():** Phương thức này được gọi khi đối tượng bắt đầu hoạt động. Nó khởi tạo các biến bằng các giá trị từ MeleeEnemy\_details.

**Update():** Phương thức này được gọi mỗi khung hình. Nó tăng bộ đếm thời gian hồi chiêu, kiểm tra xem người chơi có trong tầm nhìn hay không và thực hiện tấn công nếu có thể.

* Nếu người chơi trong tầm nhìn và thời gian hồi chiêu đã hết, thực hiện tấn công bằng cách kích hoạt hoạt ảnh "Attack1" hoặc "Attack2" dựa trên giá trị ngẫu nhiên.
* Vô hiệu hóa hoặc kích hoạt hành vi tuần tra dựa trên việc người chơi có trong tầm nhìn hay không.

**PlayerInSight():** Phương thức này kiểm tra xem người chơi có trong tầm nhìn của enemy hay không bằng cách sử dụng BoxCast là một hàm được xây dựng sẵn trong unity

**DamagePlayer():** Phương thức này gây sát thương cho người chơi khi người chơi trong tầm nhìn. Biến randomAttack dùng để cho phép ngẫu nhiên enemy gây sát thương thường hay trọng kích, có 30% khả năng gây sát thương trọng kích. Và biến randomAttack sẽ được đưa vào hàm phương thức update() để thay đổi hoạt ảnh cho phù hợp với từng loại sát thương.

**OnDrawGizmos():** Phương thức này vẽ một hình hộp trong chế độ xem Scene của Unity để trực quan hóa phạm vi phát hiện của enemy. Phương thức này dùng trong thiết kế để nhận biết tầm nhìn của enemy, không hiển thị trên phần game chính.

Phần này đảm bảo rằng enemy cận chiến có thể phát hiện người chơi, tấn công khi có thể và tuần tra khi không phát hiện thấy người chơi.

**Patroll.cs**

using UnityEngine;

public class Patroll : MonoBehaviour

{

[SerializeField] private Transform leftEdge;

[SerializeField] private Transform rightEdge;

[SerializeField] private Transform enemyTaget;

[SerializeField] private Animator anim;

[SerializeField] private Enemy\_group Enemy\_details;

private float Speed;

private float idleDuration;

private bool MovingLeft = false;

private float idleTimer;

private void Start()

{

Speed = Enemy\_details.Speed;

idleDuration = Enemy\_details.idleDuration;

}

private void Update()

{

if (MovingLeft)

{

if (enemyTaget.position.x >= leftEdge.position.x) MoveInDirection(-1);

else

{

idleTimer += Time.deltaTime;

if (idleTimer >= idleDuration) MovingLeft = !MovingLeft;

anim.SetBool("moving", false);

}

}

else

{

if (enemyTaget.position.x <= rightEdge.position.x) MoveInDirection(1);

else

{

idleTimer += Time.deltaTime;

if (idleTimer >= idleDuration) MovingLeft = !MovingLeft;

anim.SetBool("moving", false);

}

}

}

private void MoveInDirection(int direction)

{

idleTimer = 0;

anim.SetBool("moving", true);

enemyTaget.localScale = new Vector2(Mathf.Abs(enemyTaget.localScale.x) \* direction, enemyTaget.localScale.y);

enemyTaget.position = new Vector2(enemyTaget.position.x + Time.deltaTime \* direction \* Speed, enemyTaget.position.y);

}

private void OnDisable()

{

anim.SetBool("moving", false);

}

}

Đoạn mã này xử lý hành vi tuần tra của một enemy trong trò chơi Unity. Enemy di chuyển qua lại giữa hai điểm được xác định trước và dừng lại trong một khoảng thời gian cố định tại mỗi điểm trước khi quay đầu di chuyển lại.

Biến thành viên:

* Transform leftEdge: Điểm giới hạn bên trái mà enemy tuần tra đến.
* Transform rightEdge: Điểm giới hạn bên phải mà enemy tuần tra đến.
* Transform enemyTaget: Điểm mục tiêu mà enemy di chuyển tới.
* Animator anim: Thành phần Animator dùng để điều khiển hoạt ảnh của enemy.
* Enemy\_group Enemy\_details: Thông tin chi tiết về enemy, bao gồm tốc độ di chuyển và thời gian chờ.
* float Speed: Tốc độ di chuyển của enemy.
* float idleDuration: Thời gian chờ khi enemy đến một trong hai giới hạn tuần tra.
* bool MovingLeft: Biến xác định hướng di chuyển của enemy, ban đầu được đặt là di chuyển sang trái.
* float idleTimer: Bộ đếm thời gian chờ khi enemy đến một trong hai giới hạn tuần tra.

**Start():** Phương thức này được gọi khi đối tượng bắt đầu hoạt động. Nó khởi tạo các biến Speed và idleDuration bằng các giá trị từ Enemy\_details.

**Update():** Phương thức này được gọi mỗi khung hình. Nó xử lý việc di chuyển của enemy và thời gian chờ khi enemy đến một trong hai giới hạn tuần tra.

* Nếu enemy đang di chuyển sang trái và chưa đạt đến giới hạn bên trái, gọi MoveInDirection(-1) để tiếp tục di chuyển sang trái.
* Nếu enemy đang di chuyển sang phải và chưa đạt đến giới hạn bên phải, gọi MoveInDirection(1) để tiếp tục di chuyển sang phải.
* Nếu enemy đã đạt đến một trong hai giới hạn, tăng idleTimer theo thời gian trôi qua. Nếu idleTimer lớn hơn hoặc bằng idleDuration, đổi hướng di chuyển (MovingLeft = !MovingLeft) và đặt hoạt ảnh moving thành false.

**MoveInDirection(int direction):** Phương thức này xử lý việc di chuyển của enemy theo hướng được chỉ định và đặt lại bộ đếm thời gian chờ.

* Đặt lại idleTimer bằng 0.
* Đặt hoạt ảnh moving thành true.
* Điều chỉnh hướng của enemy bằng cách thay đổi localScale của enemyTaget.
* Di chuyển enemyTaget theo hướng chỉ định với tốc độ được xác định bởi Speed.

**OnDisable():** Phương thức này được gọi khi đối tượng bị vô hiệu hóa. Nó đặt hoạt ảnh moving thành false.

**Enemy\_Health.cs**

using UnityEngine;

public class Enemy\_Health : MonoBehaviour

{

[SerializeField] private Enemy\_group Enemy\_details;

[SerializeField] private Rigidbody2D rigid\_body;

private float startingHealth;

public bool dead = false;

public float currentHealth { get; private set; }

private Animator anim;

private void Awake()

{

startingHealth = Enemy\_details.startingHealth;

currentHealth = startingHealth;

anim = GetComponent<Animator>();

}

public void TakeDamage(int damage)

{

currentHealth -= damage;

if (currentHealth > startingHealth) currentHealth = startingHealth;

if (currentHealth < 0) currentHealth = 0;

if (currentHealth > 0)

{

// take hurt

anim.SetTrigger("Hurt");

}

else

{

// go dead

if (!dead)

{

anim.SetTrigger("Die");

GetComponentInParent<Patroll>().enabled = false;

GetComponent<MeleeEnemy>().enabled = false;

rigid\_body.simulated = false;

dead = true;

}

}

}

private void Update()

{

}

}

Đoạn mã này quản lý máu của enemy trong trò chơi Unity. Khi enemy nhận sát thương, mã sẽ giảm máu hiện tại của enemy, hiển thị hoạt ảnh bị thương hoặc chết, và vô hiệu hóa các thành phần điều khiển di chuyển và tấn công của enemy khi nó chết.

Biến thành viên:

* Enemy\_group Enemy\_details: Thông tin chi tiết về enemy, bao gồm máu ban đầu.
* Rigidbody2D rigid\_body: Thành phần Rigidbody2D của enemy, dùng để xử lý vật lý.
* float startingHealth: Máu ban đầu của enemy, lấy từ Enemy\_details.
* bool dead: Biến đánh dấu enemy đã chết hay chưa.
* float currentHealth: Máu hiện tại của enemy, chỉ có thể được truy cập bên ngoài lớp thông qua thuộc tính currentHealth.
* Animator anim: Thành phần Animator dùng để điều khiển hoạt ảnh của enemy.

**Awake():** Phương thức này được gọi khi đối tượng được tạo ra. Nó khởi tạo các biến startingHealth và currentHealth bằng các giá trị từ Enemy\_details, và lấy thành phần Animator.

**TakeDamage(int damage):** Phương thức này được gọi khi enemy nhận sát thương. Nó giảm máu hiện tại của enemy theo giá trị sát thương, và xử lý các hoạt ảnh tương ứng.

Phần này đảm bảo rằng enemy có thể nhận sát thương, hiển thị hoạt ảnh bị thương khi còn sống, và vô hiệu hóa các thành phần điều khiển di chuyển và tấn công khi chết.

**Enemy\_group.cs**

using UnityEngine;

public class Enemy\_group : MonoBehaviour

{

// Start is called before the first frame update

[Header ("Details")]

[SerializeField] public int damage;

[SerializeField] public int criticalDamage;

[SerializeField] public float startingHealth;

[SerializeField] public float attackCooldown;

[SerializeField] public float detectRange;

[SerializeField] public float detectDistance;

[Header ("Patrolling")]

[SerializeField] public float Speed;

[SerializeField] public float idleDuration;

}

Chứa các biến điều chỉnh thông số cho enemy, được gọi trong 3 script Patroll.cs, Enemy\_health.cs, MeleeEnemy.cs và được truyền thông qua Serialize Field của unity. Sử dụng câu lệnh dưới đây để gọi:

[SerializeField] private Enemy\_group Enemy\_details