using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class HealthScript : MonoBehaviour

{

private EnemyAnimator enemy\_Anim;

private NavMeshAgent navAgent;//navAgent nen coi lai

private EnemyController enemy\_Controller;

public float health = 100f;

public bool is\_Player, is\_Boar, is\_Cannibal;

private bool is\_Dead;

private EnemyAudio enemyAudio;

private PlayerStats player\_Stats;

void Awake()

{

if(is\_Boar || is\_Cannibal)

{

enemy\_Anim = GetComponent<EnemyAnimator>();

enemy\_Controller = GetComponent<EnemyController>();

navAgent = GetComponent<NavMeshAgent>();

// get enemy audio

enemyAudio = GetComponentInChildren<EnemyAudio>();

}

if (is\_Player)

{

player\_Stats = GetComponent<PlayerStats>();

}

}

public void ApplyDamage(float damage)

{

//if we died don't execute the rest of the code

if(is\_Dead)

return;

health -= damage;

if(is\_Player)

{

//show the stats(display the health UI value)

player\_Stats.Display\_HealthStats(health);

}

if(is\_Boar || is\_Cannibal) {

if (enemy\_Controller.Enemy\_State == EnemyState.PATROL)

{

enemy\_Controller.chase\_Distance = 100f;

}

}

if(health<=0f){

PlayerDied();

is\_Dead = true;

}

void PlayerDied()

{

if (is\_Cannibal)

{

GetComponent<Animator>().enabled = false;

GetComponent<BoxCollider>().isTrigger =false;

GetComponent<Rigidbody>().AddTorque(-transform.forward \* 50f);

enemy\_Controller.enabled=false;

navAgent.enabled=false;

enemy\_Anim.enabled=false;

StartCoroutine(DeadSound());

//EnemyManager spawn more enemies

EnemyManager.instance.EnemyDied(true);

}

if (is\_Boar) {

navAgent.velocity = Vector3.zero;

navAgent.isStopped = true;

enemy\_Controller.enabled=false ;

enemy\_Anim.Dead();

StartCoroutine(DeadSound());

//EnemyManager spawn more enemies

EnemyManager.instance.EnemyDied(false);

}

if (is\_Player)

{

GameObject[] enemies = GameObject.FindGameObjectsWithTag(Tags.ENEMY\_TAG);

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

{

enemies[i].GetComponent<EnemyController>().enabled = false;

}

//call enemy manager to stop spawning enemies

EnemyManager.instance.StopSpawning();

GetComponent<PlayerMovement>().enabled = false;

GetComponent<PlayerAttack>().enabled = false;

GetComponent<WeaponManager>().GetCurrentSelectWeapon().gameObject.SetActive(false);

}

if(tag == Tags.PLAYER\_TAG)

{

Invoke("RestartGame", 3f);

}

else

{

Invoke("TurnOffGameObject", 3f);

}

}

//player died

void RestartGame()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");

}

void TurnOffGameObject()

{

gameObject.SetActive(false);

}

IEnumerator DeadSound(){

yield return new WaitForSeconds(0.3f);

enemyAudio.Play\_DeadSound();

}

}

/\*// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

}\*/

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class HealthScript : MonoBehaviour

{

private EnemyAnimator enemy\_Anim;

private NavMeshAgent navAgent;//navAgent nen coi lai

private EnemyController enemy\_Controller;

public float health = 100f;

public bool is\_Player, is\_Boar, is\_Cannibal;

private bool is\_Dead;

private EnemyAudio enemyAudio;

private PlayerStats player\_Stats;

Trong lớp HealthScript được cung cấp trong Unity, các biến được sử dụng để quản lý sức khỏe và trạng thái của các thực thể trò chơi khác nhau như người chơi, heo rừng và ăn thịt người.

 **enemy\_Anim (EnemyAnimator)**:

* **Mục đích**: Biến này giữ tham chiếu đến thành phần EnemyAnimator. Nó có khả năng được sử dụng để điều khiển các hoạt ảnh của kẻ thù, chẳng hạn như đi bộ, tấn công và chết.

 **navAgent (NavMeshAgent)**:

* **Mục đích**: Biến này giữ tham chiếu đến thành phần NavMeshAgent, được sử dụng cho điều hướng và tìm đường. Nó cho phép kẻ thù di chuyển về phía mục tiêu, theo dõi các đường dẫn và tránh chướng ngại vật trong môi trường trò chơi.

 **enemy\_Controller (EnemyController)**:

* **Mục đích**: Biến này giữ tham chiếu đến thành phần EnemyController, có thể quản lý hành vi và trạng thái tổng thể của kẻ thù, chẳng hạn như ra quyết định và phản ứng với hành động của người chơi.

 **health**:

* **Mục đích**: Biến công khai này xác định sức khỏe của thực thể (người chơi, heo rừng, hoặc ăn thịt người). Các script khác có thể truy cập và thay đổi để tăng hoặc giảm sức khỏe của thực thể.

 **is\_Player, is\_Boar, is\_Cannibal**:

* **Mục đích**: Các cờ boolean này cho biết loại thực thể mà script này được gắn vào. Chỉ một trong những biến này nên là true tại một thời điểm. Điều này giúp script xác định các hành động hoặc hành vi cụ thể tùy thuộc vào loại thực thể (ví dụ: xử lý logic cụ thể của người chơi so với logic cụ thể của kẻ thù).

 **is\_Dead**:

* **Mục đích**: Biến boolean này theo dõi xem thực thể đã chết hay chưa. Nó có thể được sử dụng để ngăn chặn một số hành động xảy ra sau khi thực thể đã chết, chẳng hạn như di chuyển hoặc tấn công.

 **enemyAudio (EnemyAudio)**:

* **Mục đích**: Biến này giữ tham chiếu đến thành phần EnemyAudio, được sử dụng để phát âm thanh liên quan đến kẻ thù, chẳng hạn như tiếng gầm, tiếng hét hoặc âm thanh khi chết.

 **player\_Stats (PlayerStats)**:

* **Mục đích**: Biến này giữ tham chiếu đến thành phần PlayerStats, có khả năng chứa các thống kê và thuộc tính khác nhau của nhân vật người chơi, chẳng hạn như sức khỏe, sức bền và điểm kinh nghiệm. Điều này có thể được sử dụng để cập nhật hoặc truy xuất dữ liệu cụ thể của người chơi.

Kết luận

Lớp HealthScript được sử dụng để quản lý sức khỏe và trạng thái chết của các loại thực thể khác nhau trong trò chơi (người chơi, heo rừng, ăn thịt người). Tùy thuộc vào loại thực thể, nó tương tác với các thành phần khác như hoạt ảnh, điều hướng và âm thanh để cung cấp phản ứng thích hợp với các sự kiện trong trò chơi như bị tổn thương hoặc chết. Các tham chiếu đến các script và thành phần khác cho phép quản lý hành vi phức tạp một cách mô-đun và có tổ chức.

void Awake()

{

if(is\_Boar || is\_Cannibal)

{

enemy\_Anim = GetComponent<EnemyAnimator>();

enemy\_Controller = GetComponent<EnemyController>();

navAgent = GetComponent<NavMeshAgent>();

// get enemy audio

enemyAudio = GetComponentInChildren<EnemyAudio>();

}

if (is\_Player)

{

player\_Stats = GetComponent<PlayerStats>();

}

}

Hàm Awake này trong Unity được sử dụng để khởi tạo các biến và thiết lập các tham chiếu khi đối tượng chứa script được tạo ra trong game. Cụ thể, nó kiểm tra loại thực thể mà script này được gắn vào (heo rừng, ăn thịt người, hoặc người chơi) và sau đó khởi tạo các thành phần tương ứng.

 **Awake Method**:

* **Mục đích**: Awake là một hàm được Unity gọi tự động khi đối tượng được kích hoạt. Nó được sử dụng để khởi tạo bất kỳ tham chiếu hoặc biến nào cần thiết trước khi game bắt đầu chạy.

 **Kiểm tra loại thực thể (is\_Boar hoặc is\_Cannibal)**:

* **Điều kiện**: if(is\_Boar || is\_Cannibal)
  + Nếu thực thể là heo rừng (is\_Boar) hoặc ăn thịt người (is\_Cannibal), thực hiện các bước sau:

enemy\_Anim = GetComponent<EnemyAnimator>();

* + **enemy\_Anim**: Lấy tham chiếu đến thành phần EnemyAnimator để điều khiển hoạt ảnh của kẻ thù.

enemy\_Controller = GetComponent<EnemyController>();

**enemy\_Controller**: Lấy tham chiếu đến thành phần EnemyController để quản lý hành vi và trạng thái của kẻ thù.

navAgent = GetComponent<NavMeshAgent>();

**navAgent**: Lấy tham chiếu đến thành phần NavMeshAgent để điều hướng và tìm đường cho kẻ thù.

enemyAudio = GetComponentInChildren<EnemyAudio>();

**enemyAudio**: Lấy tham chiếu đến thành phần EnemyAudio để phát các âm thanh liên quan đến kẻ thù (tìm trong các thành phần con).

**Kiểm tra loại thực thể (is\_Player)**:

* **Điều kiện**: if (is\_Player)
  + Nếu thực thể là người chơi (is\_Player), thực hiện các bước sau:

player\_Stats = GetComponent<PlayerStats>();

**player\_Stats**: Lấy tham chiếu đến thành phần PlayerStats để quản lý các thống kê và thuộc tính của người chơi.

Kết luận:

Hàm Awake này khởi tạo và thiết lập các tham chiếu cần thiết cho các thành phần khác nhau dựa trên loại thực thể (người chơi, heo rừng, hoặc ăn thịt người). Điều này đảm bảo rằng tất cả các thành phần cần thiết đều được khởi tạo và sẵn sàng sử dụng khi game bắt đầu chạy.

public void ApplyDamage(float damage)

{

//if we died don't execute the rest of the code

if(is\_Dead)

return;

health -= damage;

if(is\_Player)

{

//show the stats(display the health UI value)

player\_Stats.Display\_HealthStats(health);

}

if(is\_Boar || is\_Cannibal) {

if (enemy\_Controller.Enemy\_State == EnemyState.PATROL)

{

enemy\_Controller.chase\_Distance = 100f;

}

}

if(health<=0f){

PlayerDied();

is\_Dead = true;

}

Hàm ApplyDamage trong Unity được sử dụng để xử lý việc giảm sức khỏe của một thực thể khi nó nhận sát thương. Hàm này sẽ cập nhật sức khỏe của thực thể, kiểm tra xem thực thể đã chết chưa, và thực hiện các hành động tương ứng dựa trên loại thực thể (người chơi, heo rừng, hoặc ăn thịt người).

**Kiểm tra nếu thực thể đã chết (is\_Dead)**:

* **Mục đích**: Nếu thực thể đã chết, hàm sẽ kết thúc ngay lập tức mà không thực hiện các bước còn lại.

if(is\_Dead)

return;

health -= damage;

**Giảm sức khỏe của thực thể**:

* **Mục đích**: Giảm sức khỏe của thực thể bằng giá trị sát thương nhận được.

if(is\_Player)

{

//show the stats(display the health UI value)

player\_Stats.Display\_HealthStats(health);

}

**Cập nhật giao diện người dùng nếu là người chơi (is\_Player)**:

* **Mục đích**: Nếu thực thể là người chơi, cập nhật giá trị sức khỏe trên giao diện người dùng.

if(is\_Boar || is\_Cannibal) {

if (enemy\_Controller.Enemy\_State == EnemyState.PATROL)

{

enemy\_Controller.chase\_Distance = 100f;

}

}

**Thay đổi hành vi của kẻ thù nếu là heo rừng hoặc ăn thịt người (is\_Boar hoặc is\_Cannibal)**:

* **Mục đích**: Nếu thực thể là heo rừng hoặc ăn thịt người và đang ở trạng thái tuần tra, thay đổi khoảng cách truy đuổi để bắt đầu đuổi theo mục tiêu.

if(health<=0f){

PlayerDied();

is\_Dead = true;

}

**Kiểm tra xem thực thể đã chết chưa**:

* **Mục đích**: Nếu sức khỏe của thực thể giảm xuống 0 hoặc thấp hơn, thực thể sẽ chết. Gọi hàm PlayerDied để xử lý các hành động khi chết và đặt is\_Dead thành true.

Kết luận:

Hàm ApplyDamage được sử dụng để xử lý việc nhận sát thương của thực thể, cập nhật sức khỏe, kiểm tra xem thực thể đã chết chưa, và thực hiện các hành động thích hợp dựa trên loại thực thể. Điều này bao gồm cập nhật giao diện người dùng cho người chơi, thay đổi hành vi của kẻ thù và xử lý các hành động khi thực thể chết.

void PlayerDied()

{

if (is\_Cannibal)

{

GetComponent<Animator>().enabled = false;

GetComponent<BoxCollider>().isTrigger =false;

GetComponent<Rigidbody>().AddTorque(-transform.forward \* 50f);

enemy\_Controller.enabled=false;

navAgent.enabled=false;

enemy\_Anim.enabled=false;

StartCoroutine(DeadSound());

//EnemyManager spawn more enemies

EnemyManager.instance.EnemyDied(true);

}

if (is\_Boar) {

navAgent.velocity = Vector3.zero;

navAgent.isStopped = true;

enemy\_Controller.enabled=false ;

enemy\_Anim.Dead();

StartCoroutine(DeadSound());

//EnemyManager spawn more enemies

EnemyManager.instance.EnemyDied(false);

}

if (is\_Player)

{

GameObject[] enemies = GameObject.FindGameObjectsWithTag(Tags.ENEMY\_TAG);

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

{

enemies[i].GetComponent<EnemyController>().enabled = false;

}

//call enemy manager to stop spawning enemies

EnemyManager.instance.StopSpawning();

GetComponent<PlayerMovement>().enabled = false;

GetComponent<PlayerAttack>().enabled = false;

GetComponent<WeaponManager>().GetCurrentSelectWeapon().gameObject.SetActive(false);

}

if(tag == Tags.PLAYER\_TAG)

{

Invoke("RestartGame", 3f);

}

else

{

Invoke("TurnOffGameObject", 3f);

}

}

if (is\_Cannibal)

{

GetComponent<Animator>().enabled = false;

GetComponent<BoxCollider>().isTrigger =false;

GetComponent<Rigidbody>().AddTorque(-transform.forward \* 50f);

enemy\_Controller.enabled=false;

navAgent.enabled=false;

enemy\_Anim.enabled=false;

StartCoroutine(DeadSound());

//EnemyManager spawn more enemies

EnemyManager.instance.EnemyDied(true);

}

**Kiểm tra nếu thực thể là ăn thịt người (is\_Cannibal)**:

* **Mục đích**: Thực hiện các hành động cần thiết khi ăn thịt người chết.
* **Vô hiệu hóa animator**: Dừng hoạt ảnh.
* **Thiết lập BoxCollider**: Đảm bảo collider không phải là trigger.
* **Thêm lực xoay vào Rigidbody**: Tạo hiệu ứng xoay khi chết.
* **Vô hiệu hóa điều khiển và điều hướng**: Dừng điều khiển và di chuyển.
* **Bắt đầu âm thanh chết**: Chạy coroutine để phát âm thanh.
* **Yêu cầu EnemyManager sinh thêm kẻ thù**: Thông báo cho EnemyManager rằng kẻ thù đã chết.

if (is\_Boar) {

navAgent.velocity = Vector3.zero;

navAgent.isStopped = true;

enemy\_Controller.enabled=false ;

enemy\_Anim.Dead();

StartCoroutine(DeadSound());

//EnemyManager spawn more enemies

EnemyManager.instance.EnemyDied(false);

}

**Kiểm tra nếu thực thể là heo rừng (is\_Boar)**:

* **Mục đích**: Thực hiện các hành động cần thiết khi heo rừng chết.
* **Dừng điều hướng**: Đặt vận tốc về 0 và dừng navAgent.
* **Vô hiệu hóa điều khiển**: Dừng điều khiển của heo rừng.
* **Kích hoạt hoạt ảnh chết**: Chạy hoạt ảnh chết.
* **Bắt đầu âm thanh chết**: Chạy coroutine để phát âm thanh.
* **Yêu cầu EnemyManager sinh thêm kẻ thù**: Thông báo cho EnemyManager rằng kẻ thù đã chết.

if (is\_Player)

{

GameObject[] enemies = GameObject.FindGameObjectsWithTag(Tags.ENEMY\_TAG);

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

{

enemies[i].GetComponent<EnemyController>().enabled = false;

}

//call enemy manager to stop spawning enemies

EnemyManager.instance.StopSpawning();

GetComponent<PlayerMovement>().enabled = false;

GetComponent<PlayerAttack>().enabled = false;

GetComponent<WeaponManager>().GetCurrentSelectWeapon().gameObject.SetActive(false);

}

**Kiểm tra nếu thực thể là người chơi (is\_Player)**:

* **Mục đích**: Thực hiện các hành động cần thiết khi người chơi chết.
* **Tìm tất cả các kẻ thù và vô hiệu hóa điều khiển của chúng**: Dừng tất cả các kẻ thù.
* **Yêu cầu EnemyManager dừng sinh kẻ thù**: Ngừng sinh thêm kẻ thù.
* **Vô hiệu hóa điều khiển và tấn công của người chơi**: Dừng điều khiển và tấn công của người chơi.
* **Tắt vũ khí hiện tại**: Ẩn vũ khí của người chơi.

if(tag == Tags.PLAYER\_TAG)

{

Invoke("RestartGame", 3f);

}

else

{

Invoke("TurnOffGameObject", 3f);

}

}

**Kiểm tra tag của thực thể**:

* **Mục đích**: Quyết định hành động tiếp theo dựa trên loại thực thể.
* **Nếu là người chơi (Tags.PLAYER\_TAG)**: Khởi động lại trò chơi sau 3 giây.
* **Nếu là kẻ thù**: Tắt đối tượng sau 3 giây.

Kết luận:

Hàm PlayerDied xử lý các hành động khi thực thể chết, bao gồm vô hiệu hóa các thành phần liên quan, kích hoạt hiệu ứng chết, và quản lý việc sinh thêm kẻ thù. Nó đảm bảo rằng trò chơi phản ứng đúng cách với cái chết của thực thể, tạo ra trải nghiệm người chơi nhất quán và logic.

//player died

void RestartGame()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");

}

void TurnOffGameObject()

{

gameObject.SetActive(false);

}

IEnumerator DeadSound(){

yield return new WaitForSeconds(0.3f);

enemyAudio.Play\_DeadSound();

}

Các hàm và coroutine được thêm vào trong đoạn mã này cung cấp các chức năng bổ sung cho việc xử lý sự kiện khi một thực thể chết

void RestartGame()

{

UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");

}

**Giải thích chi tiết về các hàm:**

1. **RestartGame**:
   * **Mục đích**: Khởi động lại trò chơi bằng cách tải lại cảnh hiện tại hoặc một cảnh cụ thể.
   * **Chi tiết**: Hàm này sử dụng UnityEngine.SceneManagement.SceneManager.LoadScene để tải lại cảnh có tên "SampleScene".

void TurnOffGameObject()

{

gameObject.SetActive(false);

}

**TurnOffGameObject**:

* **Mục đích**: Vô hiệu hóa đối tượng hiện tại trong trò chơi.
* **Chi tiết**: Hàm này gọi gameObject.SetActive(false) để tắt đối tượng hiện tại, khiến nó không còn xuất hiện hoặc hoạt động trong cảnh.

IEnumerator DeadSound(){

yield return new WaitForSeconds(0.3f);

enemyAudio.Play\_DeadSound();

}

**DeadSound (Coroutine)**:

* **Mục đích**: Phát âm thanh khi thực thể chết sau một khoảng thời gian trễ.
* **Chi tiết**: Coroutine này sử dụng yield return new WaitForSeconds(0.3f) để đợi 0.3 giây trước khi gọi hàm enemyAudio.Play\_DeadSound() để phát âm thanh chết.

Tóm tắt:

 **RestartGame**: Khởi động lại trò chơi bằng cách tải lại cảnh có tên "SampleScene".

 **TurnOffGameObject**: Vô hiệu hóa đối tượng hiện tại để nó không còn hoạt động trong cảnh.

 **DeadSound**: Phát âm thanh khi thực thể chết sau một khoảng thời gian trễ 0.3 giây.

Kết luận:

Các hàm này thêm vào các hành động bổ sung để xử lý sự kiện khi một thực thể chết. RestartGame khởi động lại trò chơi, TurnOffGameObject vô hiệu hóa đối tượng hiện tại, và DeadSound phát âm thanh chết với một khoảng thời gian trễ nhỏ. Những hàm này được thiết kế để tạo ra các hiệu ứng và hành vi thích hợp trong trò chơi khi một thực thể gặp sự kiện chết.