using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public enum EnemyState

{

PATROL,

CHASE,

ATTACK

}

public class EnemyController : MonoBehaviour

{

private EnemyAnimator enemy\_Anim;

private NavMeshAgent navAgent;

private EnemyState enemy\_State;

public float walk\_Speed = 0.5f;

public float run\_Speed = 4f;

public float chase\_Distance = 7f;

private float current\_Chase\_Distance;

public float attack\_Distance = 1.8f;

public float chase\_After\_Attack\_Distance = 2f;

public float patrol\_Radius\_Min = 20f, patrol\_Radius\_Max = 60f;

public float patrol\_For\_This\_Time = 15f;

public float patrol\_Timer;

public float wait\_Before\_Attack = 2f;

private float attack\_Timer;

private Transform target;

public GameObject attack\_Point;

private EnemyAudio enemy\_Audio;

void Awake()

{

enemy\_Anim = GetComponent<EnemyAnimator>();

navAgent = GetComponent<NavMeshAgent>();

target = GameObject.FindWithTag(Tags.PLAYER\_TAG).transform;

enemy\_Audio = GetComponentInChildren<EnemyAudio>();

}

// Start is called before the first frame update

void Start()

{

enemy\_State = EnemyState.PATROL;

patrol\_Timer = patrol\_For\_This\_Time;

//when the enemy first gets to the player

//attack right away

attack\_Timer = wait\_Before\_Attack;

//memorize the value of chase distance

//so that we can put it back

current\_Chase\_Distance = chase\_Distance;

}

// Update is called once per frame

void Update()

{

if (enemy\_State == EnemyState.PATROL)

{

Patrol();

}

if (enemy\_State == EnemyState.CHASE)

{

Chase();

}

if (enemy\_State == EnemyState.ATTACK)

{

Attack();

}

}

void Patrol()

{

// tell nav agent that he can move

navAgent.isStopped = false;

navAgent.speed = walk\_Speed;

//add to the patrol timer

patrol\_Timer += Time.deltaTime;

if (patrol\_Timer > patrol\_For\_This\_Time)

{

SetNewRandomDestination();

patrol\_Timer = 0f;

}

if (navAgent.velocity.sqrMagnitude > 0)

{

enemy\_Anim.Walk(true);

}

else

{

enemy\_Anim.Walk(false);

}

//test the distance of player and enemy

if (Vector3.Distance(transform.position, target.position) <= chase\_Distance)

{

enemy\_Anim.Walk(false);

enemy\_State = EnemyState.CHASE;

//play spotted audio

enemy\_Audio.Play\_ScreamSound();

}

}

void Chase()

{

//enable enemy to move again

navAgent.isStopped = false;

navAgent.speed = run\_Speed;

//set the player's position as the destination

//because we are chasing(running toward ) the player

navAgent.SetDestination(target.position);

if (navAgent.velocity.sqrMagnitude > 0)

{

enemy\_Anim.Run(true);

}

else

{

enemy\_Anim.Run(false);

}

//if distance of player and enemy is short enough(less than attack\_Distance => attack

if (Vector3.Distance(transform.position, target.position) <= attack\_Distance)

{

enemy\_Anim.Run(false);

enemy\_Anim.Walk(false);

enemy\_State = EnemyState.ATTACK;

if (chase\_Distance != current\_Chase\_Distance)

{

chase\_Distance = current\_Chase\_Distance;

}

}

else if (Vector3.Distance(transform.position, target.position) > chase\_Distance)

{

//player run away from enemy

//stop running

enemy\_Anim.Run(false);

//get back to patrol

enemy\_State = EnemyState.PATROL;

//reset patrol timmer

//can calculate the new patrol destination right away

patrol\_Timer = patrol\_For\_This\_Time;

if (chase\_Distance != current\_Chase\_Distance)

{

chase\_Distance = current\_Chase\_Distance;

}

}//else

}

void Attack()

{

navAgent.velocity = Vector3.zero;

navAgent.isStopped = true;

attack\_Timer += Time.deltaTime;

if(attack\_Timer>wait\_Before\_Attack)

{

enemy\_Anim.Attack();

attack\_Timer = 0f;

//play Attack sound

enemy\_Audio.Play\_AttackSound();

}

if(Vector3.Distance(transform.position,target.position) > attack\_Distance+chase\_After\_Attack\_Distance) {

enemy\_State=EnemyState.CHASE;

}

}

void SetNewRandomDestination()

{

float rand\_Radius = Random.Range(patrol\_Radius\_Min, patrol\_Radius\_Max);

Vector3 randDir = Random.insideUnitSphere \* rand\_Radius;

randDir += transform.position;

NavMeshHit navHit;

NavMesh.SamplePosition(randDir, out navHit, rand\_Radius, -1);

navAgent.SetDestination(navHit.position);

}

void Turn\_On\_AttackPoint()

{

attack\_Point.SetActive(true);

}

void Turn\_Off\_AttackPoint()

{

if (attack\_Point.activeInHierarchy)

{

attack\_Point.SetActive(false);

}

}

public EnemyState Enemy\_State

{

get; set;

}

}

public enum EnemyState

{

PATROL,

CHASE,

ATTACK

}

Enum EnemyState định nghĩa một tập hợp các hằng số có tên đại diện cho các trạng thái khác nhau mà một nhân vật kẻ thù có thể ở trong một trò chơi. Điều này thường được sử dụng trong phát triển trò chơi, đặc biệt là trong logic AI (trí tuệ nhân tạo) của nhân vật kẻ thù. Mỗi trạng thái đại diện cho một hành vi hoặc tập hợp các hành động mà kẻ thù có thể thực hiện.

**PATROL (Tuần tra)**: Trạng thái này thường có nghĩa là kẻ thù đang di chuyển quanh một con đường hoặc khu vực được định sẵn, tìm kiếm người chơi hoặc đơn giản là duy trì sự hiện diện trong thế giới trò chơi. Trong trạng thái này, kẻ thù không chủ động đuổi theo người chơi.

**CHASE (Đuổi theo)**: Khi kẻ thù phát hiện người chơi, nó chuyển sang trạng thái CHASE. Trong trạng thái này, kẻ thù chủ động đuổi theo người chơi, cố gắng tiếp cận đủ gần để tấn công.

**ATTACK (Tấn công)**: Khi kẻ thù đủ gần người chơi, nó chuyển sang trạng thái ATTACK. Trong trạng thái này, kẻ thù sẽ thực hiện các hành động tấn công của mình, chẳng hạn như vung vũ khí, bắn, hoặc bất kỳ hình thức tấn công nào khác cụ thể cho trò chơi.

public class EnemyController : MonoBehaviour

{

private EnemyAnimator enemy\_Anim;

private NavMeshAgent navAgent;

private EnemyState enemy\_State;

public float walk\_Speed = 0.5f;

public float run\_Speed = 4f;

public float chase\_Distance = 7f;

private float current\_Chase\_Distance;

public float attack\_Distance = 1.8f;

public float chase\_After\_Attack\_Distance = 2f;

public float patrol\_Radius\_Min = 20f, patrol\_Radius\_Max = 60f;

public float patrol\_For\_This\_Time = 15f;

public float patrol\_Timer;

public float wait\_Before\_Attack = 2f;

private float attack\_Timer;

private Transform target;

public GameObject attack\_Point;

private EnemyAudio enemy\_Audio;

Các biến trong lớp EnemyController này đều phục vụ các mục đích khác nhau để điều khiển hành vi của kẻ thù trong trò chơi.

 **private EnemyAnimator enemy\_Anim;**

* Biến này đại diện cho đối tượng EnemyAnimator, có thể chứa logic để điều khiển các hoạt ảnh (animation) của kẻ thù.

 **private NavMeshAgent navAgent;**

* Biến này đại diện cho đối tượng NavMeshAgent, được sử dụng để điều khiển chuyển động của kẻ thù trong thế giới trò chơi sử dụng tính năng điều hướng của Unity.

 **private EnemyState enemy\_State;**

* Biến này lưu trữ trạng thái hiện tại của kẻ thù, sử dụng enum EnemyState đã được định nghĩa trước đó (PATROL, CHASE, ATTACK).

 **public float walk\_Speed = 0.5f;**

* Tốc độ di chuyển của kẻ thù khi tuần tra.

 **public float run\_Speed = 4f;**

* Tốc độ di chuyển của kẻ thù khi đuổi theo người chơi.

 **public float chase\_Distance = 7f;**

* Khoảng cách mà kẻ thù bắt đầu đuổi theo người chơi.

 **private float current\_Chase\_Distance;**

* Biến này lưu trữ khoảng cách hiện tại mà kẻ thù sẽ tiếp tục đuổi theo người chơi sau khi phát hiện ra họ.

 **public float attack\_Distance = 1.8f;**

* Khoảng cách mà kẻ thù bắt đầu tấn công người chơi.

 **public float chase\_After\_Attack\_Distance = 2f;**

* Khoảng cách mà kẻ thù sẽ quay lại đuổi theo người chơi sau khi thực hiện một đợt tấn công.

 **public float patrol\_Radius\_Min = 20f, patrol\_Radius\_Max = 60f;**

* Khoảng cách tối thiểu và tối đa mà kẻ thù có thể tuần tra xung quanh vị trí ban đầu.

 **public float patrol\_For\_This\_Time = 15f;**

* Thời gian mà kẻ thù sẽ tuần tra trước khi chuyển sang hành vi khác.

 **public float patrol\_Timer;**

* Biến này được sử dụng để đếm ngược thời gian tuần tra còn lại.

 **public float wait\_Before\_Attack = 2f;**

* Thời gian chờ trước khi kẻ thù thực hiện một đợt tấn công khác.

 **private float attack\_Timer;**

* Biến này đếm ngược thời gian còn lại trước khi kẻ thù có thể tấn công lần nữa.

 **private Transform target;**

* Biến này lưu trữ vị trí của mục tiêu (thường là người chơi).

 **public GameObject attack\_Point;**

* Biến này đại diện cho điểm tấn công của kẻ thù, có thể là nơi mà các vụ tấn công của kẻ thù sẽ được xuất phát.

 **private EnemyAudio enemy\_Audio;**

* Biến này đại diện cho đối tượng EnemyAudio, có thể chứa các âm thanh được phát khi kẻ thù thực hiện các hành động như tấn công hoặc di chuyển.

void Awake()

{

enemy\_Anim = GetComponent<EnemyAnimator>();

navAgent = GetComponent<NavMeshAgent>();

target = GameObject.FindWithTag(Tags.PLAYER\_TAG).transform;

enemy\_Audio = GetComponentInChildren<EnemyAudio>();

}

Hàm Awake trong lớp EnemyController này được sử dụng để khởi tạo các biến và thiết lập các tham chiếu cần thiết khi đối tượng kẻ thù được tạo ra trong trò chơi.

 **enemy\_Anim = GetComponent<EnemyAnimator>();**

* Lệnh này tìm và gán đối tượng EnemyAnimator liên kết với đối tượng kẻ thù hiện tại. EnemyAnimator chứa các logic để điều khiển hoạt ảnh của kẻ thù.

 **navAgent = GetComponent<NavMeshAgent>();**

* Lệnh này tìm và gán đối tượng NavMeshAgent liên kết với đối tượng kẻ thù hiện tại. NavMeshAgent được sử dụng để điều khiển chuyển động của kẻ thù sử dụng tính năng điều hướng của Unity.

 **target = GameObject.FindWithTag(Tags.PLAYER\_TAG).transform;**

* Lệnh này tìm đối tượng trong trò chơi có tag là PLAYER\_TAG (đại diện cho người chơi) và gán tham chiếu tới transform của đối tượng đó cho biến target. Điều này cho phép kẻ thù biết vị trí của người chơi trong thế giới trò chơi.

 **enemy\_Audio = GetComponentInChildren<EnemyAudio>();**

* Lệnh này tìm và gán đối tượng EnemyAudio nằm trong các con của đối tượng kẻ thù hiện tại. EnemyAudio chứa các âm thanh được phát ra khi kẻ thù thực hiện các hành động như tấn công hoặc di chuyển.

// Start is called before the first frame update

void Start()

{

enemy\_State = EnemyState.PATROL;

patrol\_Timer = patrol\_For\_This\_Time;

//when the enemy first gets to the player

//attack right away

attack\_Timer = wait\_Before\_Attack;

//memorize the value of chase distance

//so that we can put it back

current\_Chase\_Distance = chase\_Distance;

}

Hàm Start() này là một phần của một script trong Unity, thường dùng cho việc khởi tạo các giá trị khi một đối tượng (object) được tạo hoặc khi bắt đầu trò chơi. Cụ thể, hàm này được dùng để khởi tạo trạng thái và các biến cho một kẻ thù (enemy) trong game.

1. enemy\_State = EnemyState.PATROL;
   * Thiết lập trạng thái ban đầu của kẻ thù là PATROL (tuần tra). Điều này có nghĩa là kẻ thù sẽ bắt đầu di chuyển tuần tra khi game bắt đầu.
2. patrol\_Timer = patrol\_For\_This\_Time;
   * Khởi tạo bộ đếm thời gian cho chế độ tuần tra bằng cách gán giá trị của patrol\_For\_This\_Time vào patrol\_Timer. Điều này giúp kẻ thù biết trong bao lâu nó nên tuần tra trước khi thay đổi hành vi.
3. attack\_Timer = wait\_Before\_Attack;
   * Khởi tạo bộ đếm thời gian tấn công. Khi kẻ thù gặp người chơi lần đầu, nó sẽ tấn công ngay lập tức. wait\_Before\_Attack là khoảng thời gian chờ trước khi kẻ thù thực hiện tấn công tiếp theo.
4. current\_Chase\_Distance = chase\_Distance;
   * Lưu trữ giá trị khoảng cách đuổi theo ban đầu vào current\_Chase\_Distance. Điều này giúp sau này có thể khôi phục lại khoảng cách đuổi theo nếu có thay đổi trong gameplay.

Nhìn chung, hàm Start() được dùng để thiết lập trạng thái ban đầu và các giá trị cho kẻ thù, đảm bảo rằng nó bắt đầu với các hành vi đúng và có thể tương tác với người chơi theo kịch bản định trước.

void Patrol()

{

// tell nav agent that he can move

navAgent.isStopped = false;

navAgent.speed = walk\_Speed;

//add to the patrol timer

patrol\_Timer += Time.deltaTime;

if (patrol\_Timer > patrol\_For\_This\_Time)

{

SetNewRandomDestination();

patrol\_Timer = 0f;

}

if (navAgent.velocity.sqrMagnitude > 0)

{

enemy\_Anim.Walk(true);

}

else

{

enemy\_Anim.Walk(false);

}

//test the distance of player and enemy

if (Vector3.Distance(transform.position, target.position) <= chase\_Distance)

{

enemy\_Anim.Walk(false);

enemy\_State = EnemyState.CHASE;

//play spotted audio

enemy\_Audio.Play\_ScreamSound();

}

}

Hàm Patrol() trong đoạn mã này là một phần của hành vi AI cho kẻ thù trong trò chơi. Chức năng chính của hàm này là điều khiển kẻ thù di chuyển tuần tra (patrol) và chuyển sang trạng thái đuổi theo (chase) nếu người chơi (player) nằm trong khoảng cách cho phép

Kích hoạt di chuyển của nav agent:

// tell nav agent that he can move

navAgent.isStopped = false;

navAgent.speed = walk\_Speed;

Thiết lập cho nav agent (tác nhân điều hướng) của kẻ thù có thể di chuyển và đặt tốc độ di chuyển là walk\_Speed.

Cập nhật bộ đếm thời gian tuần tra:

patrol\_Timer += Time.deltaTime;

Tăng giá trị patrol\_Timer theo thời gian thực (Time.deltaTime là thời gian đã trôi qua kể từ khung hình trước đó).

Đặt điểm đến tuần tra mới:

if (patrol\_Timer > patrol\_For\_This\_Time)

{

SetNewRandomDestination();

patrol\_Timer = 0f;

}

Nếu bộ đếm thời gian tuần tra vượt quá giá trị patrol\_For\_This\_Time, gọi hàm SetNewRandomDestination() để thiết lập một điểm đến ngẫu nhiên mới cho kẻ thù, sau đó đặt lại patrol\_Timer về 0.

Cập nhật hoạt ảnh đi bộ:

if (navAgent.velocity.sqrMagnitude > 0)

{

enemy\_Anim.Walk(true);

}

else

{

enemy\_Anim.Walk(false);

}

Kiểm tra vận tốc của nav agent. Nếu vận tốc lớn hơn 0, kích hoạt hoạt ảnh đi bộ (Walk(true)). Nếu không, dừng hoạt ảnh đi bộ (Walk(false)).

**Kiểm tra khoảng cách giữa kẻ thù và người chơi:**

if (Vector3.Distance(transform.position, target.position) <= chase\_Distance)

{

enemy\_Anim.Walk(false);

enemy\_State = EnemyState.CHASE;

//play spotted audio

enemy\_Audio.Play\_ScreamSound();

}

}

Nếu khoảng cách giữa kẻ thù và người chơi nhỏ hơn hoặc bằng chase\_Distance, dừng hoạt ảnh đi bộ (Walk(false)), chuyển trạng thái kẻ thù thành CHASE, và phát âm thanh báo hiệu (Play\_ScreamSound()).

Kết luận:

hàm Patrol() điều khiển kẻ thù tuần tra trong game. Nó di chuyển kẻ thù, cập nhật hoạt ảnh đi bộ, và kiểm tra khoảng cách với người chơi để quyết định khi nào bắt đầu đuổi theo.

void Chase()

{

//enable enemy to move again

navAgent.isStopped = false;

navAgent.speed = run\_Speed;

//set the player's position as the destination

//because we are chasing(running toward ) the player

navAgent.SetDestination(target.position);

if (navAgent.velocity.sqrMagnitude > 0)

{

enemy\_Anim.Run(true);

}

else

{

enemy\_Anim.Run(false);

}

//if distance of player and enemy is short enough(less than attack\_Distance => attack

if (Vector3.Distance(transform.position, target.position) <= attack\_Distance)

{

enemy\_Anim.Run(false);

enemy\_Anim.Walk(false);

enemy\_State = EnemyState.ATTACK;

if (chase\_Distance != current\_Chase\_Distance)

{

chase\_Distance = current\_Chase\_Distance;

}

}

else if (Vector3.Distance(transform.position, target.position) > chase\_Distance)

{

//player run away from enemy

//stop running

enemy\_Anim.Run(false);

//get back to patrol

enemy\_State = EnemyState.PATROL;

//reset patrol timmer

//can calculate the new patrol destination right away

patrol\_Timer = patrol\_For\_This\_Time;

if (chase\_Distance != current\_Chase\_Distance)

{

chase\_Distance = current\_Chase\_Distance;

}

}//else

}

Hàm Chase() trong đoạn mã này là một phần của hành vi AI cho kẻ thù trong trò chơi. Chức năng chính của hàm này là điều khiển kẻ thù đuổi theo người chơi.

//enable enemy to move again

navAgent.isStopped = false;

navAgent.speed = run\_Speed;

Thiết lập cho nav agent (tác nhân điều hướng) của kẻ thù có thể di chuyển và đặt tốc độ di chuyển là run\_Speed.

//set the player's position as the destination

//because we are chasing(running toward ) the player

navAgent.SetDestination(target.position);

Đặt điểm đến của nav agent là vị trí hiện tại của người chơi để kẻ thù đuổi theo người chơi.

if (navAgent.velocity.sqrMagnitude > 0)

{

enemy\_Anim.Run(true);

}

else

{

enemy\_Anim.Run(false);

}

Kiểm tra vận tốc của nav agent. Nếu vận tốc lớn hơn 0, kích hoạt hoạt ảnh chạy (Run(true)). Nếu không, dừng hoạt ảnh chạy (Run(false)).

//if distance of player and enemy is short enough(less than attack\_Distance => attack

if (Vector3.Distance(transform.position, target.position) <= attack\_Distance)

{

enemy\_Anim.Run(false);

enemy\_Anim.Walk(false);

enemy\_State = EnemyState.ATTACK;

if (chase\_Distance != current\_Chase\_Distance)

{

chase\_Distance = current\_Chase\_Distance;

}

}

Nếu khoảng cách giữa kẻ thù và người chơi nhỏ hơn hoặc bằng attack\_Distance, dừng các hoạt ảnh chạy và đi bộ (Run(false) và Walk(false)), chuyển trạng thái kẻ thù thành ATTACK.

Nếu chase\_Distance đã thay đổi, khôi phục lại giá trị ban đầu của chase\_Distance.

else if (Vector3.Distance(transform.position, target.position) > chase\_Distance)

{

//player run away from enemy

//stop running

enemy\_Anim.Run(false);

//get back to patrol

enemy\_State = EnemyState.PATROL;

//reset patrol timmer

//can calculate the new patrol destination right away

patrol\_Timer = patrol\_For\_This\_Time;

if (chase\_Distance != current\_Chase\_Distance)

{

chase\_Distance = current\_Chase\_Distance;

}

}//else

}

Nếu khoảng cách giữa kẻ thù và người chơi lớn hơn chase\_Distance, dừng hoạt ảnh chạy (Run(false)), chuyển trạng thái kẻ thù về PATROL (tuần tra), và đặt lại patrol\_Timer để có thể tính toán điểm đến tuần tra mới ngay lập tức.

Nếu chase\_Distance đã thay đổi, khôi phục lại giá trị ban đầu của chase\_Distance.

Kết luận:

hàm Chase() điều khiển kẻ thù đuổi theo người chơi, cập nhật hoạt ảnh tương ứng, và quyết định khi nào kẻ thù sẽ tấn công người chơi hoặc quay trở lại tuần tra nếu người chơi chạy xa khỏi kẻ thù.

void Attack()

{

navAgent.velocity = Vector3.zero;

navAgent.isStopped = true;

attack\_Timer += Time.deltaTime;

if(attack\_Timer>wait\_Before\_Attack)

{

enemy\_Anim.Attack();

attack\_Timer = 0f;

//play Attack sound

enemy\_Audio.Play\_AttackSound();

}

if(Vector3.Distance(transform.position,target.position) > attack\_Distance+chase\_After\_Attack\_Distance) {

enemy\_State=EnemyState.CHASE;

}

}

Hàm Attack() trong đoạn mã này điều khiển hành vi tấn công của kẻ thù trong trò chơi.

Dừng di chuyển của nav agent:

navAgent.velocity = Vector3.zero;

navAgent.isStopped = true;

Đặt vận tốc của nav agent về 0 và dừng nav agent, khiến kẻ thù dừng lại khi chuẩn bị tấn công.

attack\_Timer += Time.deltaTime;

Tăng giá trị attack\_Timer theo thời gian thực (Time.deltaTime là thời gian đã trôi qua kể từ khung hình trước đó).

Thực hiện tấn công nếu đủ thời gian chờ:

if(attack\_Timer>wait\_Before\_Attack)

{

enemy\_Anim.Attack();

attack\_Timer = 0f;

//play Attack sound

enemy\_Audio.Play\_AttackSound();

}

Nếu attack\_Timer vượt quá giá trị wait\_Before\_Attack, kích hoạt hoạt ảnh tấn công (Attack()) và phát âm thanh tấn công (Play\_AttackSound()). Sau đó, đặt lại attack\_Timer về 0 để chuẩn bị cho lần tấn công tiếp theo.

if(Vector3.Distance(transform.position,target.position) > attack\_Distance+chase\_After\_Attack\_Distance) {

enemy\_State=EnemyState.CHASE;

}

}

Nếu khoảng cách giữa kẻ thù và người chơi lớn hơn attack\_Distance cộng với chase\_After\_Attack\_Distance, chuyển trạng thái của kẻ thù về CHASE. Điều này có nghĩa là nếu người chơi di chuyển ra xa ngoài khoảng cách tấn công, kẻ thù sẽ bắt đầu đuổi theo lại.

Kết luận:

hàm Attack() điều khiển hành vi tấn công của kẻ thù. Kẻ thù sẽ dừng lại khi tấn công, thực hiện tấn công nếu đã chờ đủ thời gian, phát âm thanh tấn công, và kiểm tra khoảng cách để quyết định khi nào nên chuyển sang trạng thái đuổi theo nếu người chơi di chuyển ra khỏi phạm vi tấn công.

void SetNewRandomDestination()

{

float rand\_Radius = Random.Range(patrol\_Radius\_Min, patrol\_Radius\_Max);

Vector3 randDir = Random.insideUnitSphere \* rand\_Radius;

randDir += transform.position;

NavMeshHit navHit;

NavMesh.SamplePosition(randDir, out navHit, rand\_Radius, -1);

navAgent.SetDestination(navHit.position);

}

Hàm SetNewRandomDestination() trong đoạn mã này được sử dụng để thiết lập một điểm đến ngẫu nhiên cho kẻ thù trong trò chơi khi nó đang tuần tra.

**Tạo một bán kính ngẫu nhiên:**

float rand\_Radius = Random.Range(patrol\_Radius\_Min, patrol\_Radius\_Max);

Tạo một giá trị bán kính ngẫu nhiên trong khoảng từ patrol\_Radius\_Min đến patrol\_Radius\_Max. Giá trị này xác định khoảng cách tối đa mà kẻ thù có thể di chuyển tới từ vị trí hiện tại để tuần tra.

**Tạo một hướng ngẫu nhiên trong phạm vi bán kính:**

Vector3 randDir = Random.insideUnitSphere \* rand\_Radius;

randDir += transform.position;

Tạo một hướng ngẫu nhiên trong hình cầu đơn vị (unit sphere) và nhân nó với rand\_Radius để có được vị trí ngẫu nhiên trong bán kính đó. Sau đó, cộng vị trí hiện tại của kẻ thù (transform.position) vào để tính toán vị trí đích cuối cùng.

**Lấy vị trí hợp lệ trên NavMesh:**

NavMeshHit navHit;

NavMesh.SamplePosition(randDir, out navHit, rand\_Radius, -1);

Sử dụng NavMesh.SamplePosition để lấy một vị trí hợp lệ trên NavMesh gần với vị trí ngẫu nhiên đã tính toán (randDir). Kết quả được lưu vào navHit. Tham số -1 ở cuối hàm NavMesh.SamplePosition cho phép tìm kiếm trên tất cả các vùng NavMesh.

navAgent.SetDestination(navHit.position);

Đặt vị trí hợp lệ tìm được (navHit.position) làm điểm đến mới cho nav agent của kẻ thù.

Kết luận:

hàm SetNewRandomDestination() thực hiện các bước sau:

* Tạo một bán kính ngẫu nhiên để xác định phạm vi tuần tra.
* Tạo một hướng ngẫu nhiên trong phạm vi đó và tính toán vị trí ngẫu nhiên.
* Sử dụng NavMesh để đảm bảo vị trí ngẫu nhiên nằm trên đường di chuyển hợp lệ.
* Thiết lập điểm đến mới cho kẻ thù trên NavMesh để tiếp tục tuần tra.

Hàm này đảm bảo rằng kẻ thù có thể di chuyển tới các vị trí ngẫu nhiên hợp lệ trong quá trình tuần tra, giúp cho hành vi của kẻ thù trở nên đa dạng và khó đoán hơn.

void Turn\_On\_AttackPoint()

{

attack\_Point.SetActive(true);

}

void Turn\_Off\_AttackPoint()

{

if (attack\_Point.activeInHierarchy)

{

attack\_Point.SetActive(false);

}

}

public EnemyState Enemy\_State

{

get; set;

}

}

Các phương thức và thuộc tính trong đoạn mã này được sử dụng để quản lý trạng thái tấn công của kẻ thù trong trò chơi.

**Phương thức Turn\_On\_AttackPoint()**

void Turn\_On\_AttackPoint()

{

attack\_Point.SetActive(true);

}

Phương thức này được sử dụng để kích hoạt điểm tấn công của kẻ thù (attack\_Point). Thường thì điểm tấn công sẽ là một GameObject (ví dụ: một vũ khí hay một phần cơ thể đặc biệt) được kích hoạt khi kẻ thù muốn thực hiện hành động tấn công.

**Phương thức Turn\_Off\_AttackPoint()**

void Turn\_Off\_AttackPoint()

{

if (attack\_Point.activeInHierarchy)

{

attack\_Point.SetActive(false);

}

}

Phương thức này kiểm tra xem điểm tấn công (attack\_Point) có đang hoạt động không (activeInHierarchy). Nếu có, nó sẽ tắt điểm tấn công. Điều này đảm bảo rằng sau khi kết thúc hành động tấn công, điểm tấn công sẽ được vô hiệu hóa để không gây ảnh hưởng đến gameplay hoặc hành vi khác của kẻ thù.

**Thuộc tính Enemy\_State**

public EnemyState Enemy\_State

{

get; set;

}

Đây là một thuộc tính công cộng (public property) có kiểu dữ liệu là EnemyState. Trong C#, EnemyState có thể là một enum định nghĩa các trạng thái của kẻ thù như PATROL, CHASE, ATTACK, và có thể thêm các trạng thái khác nếu cần. Điều quan trọng là thuộc tính này cung cấp một cách để lấy và thiết lập trạng thái của kẻ thù từ bên ngoài, cho phép các phần khác của mã có thể kiểm soát và phản ứng đáng phù hợp với trạng thái hiện tại của kẻ thù.

Kết luận:

Các phương thức Turn\_On\_AttackPoint() và Turn\_Off\_AttackPoint() dùng để quản lý và điều khiển việc hiển thị và ẩn điểm tấn công của kẻ thù, trong khi thuộc tính Enemy\_State cung cấp cơ chế để theo dõi và thay đổi trạng thái của kẻ thù trong game. Điều này làm cho hành vi của kẻ thù trở nên linh hoạt và có thể điều khiển được dễ dàng trong các tình huống khác nhau của trò chơi.