using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyManager : MonoBehaviour

{

public static EnemyManager instance;

[SerializeField]

private GameObject boar\_Prefab, cannibal\_Prefab;

public Transform[] cannibal\_SpawnPoints, boar\_SpawnPoint;

[SerializeField]

private int cannibal\_Enemy\_Count,boar\_Enemy\_Count;

private int initial\_Cannibal\_Count, initial\_Boar\_Count;

public float wait\_Before\_Spawn\_Enemies\_Time = 10f;

void Awake()

{

MakeInstance();

}

// Start is called before the first frame update

void Start()

{

initial\_Cannibal\_Count = cannibal\_Enemy\_Count;

initial\_Boar\_Count = cannibal\_Enemy\_Count;

SpawnEnemies();

StartCoroutine("CheckToSpawnEnemies");

}

void MakeInstance()

{

if (instance == null)

{

instance=this;

}

}

void SpawnEnemies()

{

SpawnCannibal();

SpawnBoars();

}

void SpawnCannibal()

{

int index = 0;

for(int i=0;i<cannibal\_Enemy\_Count;i++)

{

if(index >= cannibal\_SpawnPoints.Length)

{

index = 0;

}

Instantiate(cannibal\_Prefab, cannibal\_SpawnPoints[index].position,Quaternion.identity);

index++;

}

cannibal\_Enemy\_Count = 0;

}

void SpawnBoars()

{

int index = 0;

for (int i = 0; i < boar\_Enemy\_Count; i++)

{

if (index >= boar\_SpawnPoint.Length)

{

index = 0;

}

Instantiate(boar\_Prefab, boar\_SpawnPoint[index].position, Quaternion.identity);

index++;

}

boar\_Enemy\_Count = 0;

}

IEnumerator CheckToSpawnEnemies()

{

yield return new WaitForSeconds(wait\_Before\_Spawn\_Enemies\_Time);

SpawnCannibal();

SpawnBoars();

StartCoroutine("CheckToSpawnEnemies");

}

public void EnemyDied(bool cannibal)

{

if (cannibal)

{

cannibal\_Enemy\_Count++;

if(cannibal\_Enemy\_Count > initial\_Cannibal\_Count)

{

cannibal\_Enemy\_Count =initial\_Cannibal\_Count;

}

}

else

{

boar\_Enemy\_Count++;

if(boar\_Enemy\_Count > initial\_Cannibal\_Count)

{

boar\_Enemy\_Count=initial\_Cannibal\_Count;

}

}

}

public void StopSpawning()

{

StopCoroutine("CheckToSpawnEnemies");

}

// Update is called once per frame

void Update()

{

}

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class EnemyManager : MonoBehaviour

{

public static EnemyManager instance;

[SerializeField]

private GameObject boar\_Prefab, cannibal\_Prefab;

public Transform[] cannibal\_SpawnPoints, boar\_SpawnPoint;

[SerializeField]

private int cannibal\_Enemy\_Count,boar\_Enemy\_Count;

private int initial\_Cannibal\_Count, initial\_Boar\_Count;

public float wait\_Before\_Spawn\_Enemies\_Time = 10f;

Các biến được sử dụng để quản lý kẻ thù trong trò chơi Unity. Dưới đây là giải thích chi tiết về mục đích của từng biến:

1. **public static EnemyManager instance**: Đây là một biến tĩnh, dùng để tạo ra một instance duy nhất của lớp EnemyManager. Điều này giúp dễ dàng truy cập và quản lý các đối tượng kẻ thù từ các lớp khác trong game.
2. **[SerializeField] private GameObject boar\_Prefab, cannibal\_Prefab**: Hai biến này là tham chiếu đến prefab (mẫu) của kẻ thù "boar" và "cannibal". Prefab là các đối tượng mẫu trong Unity, dùng để tạo ra các instance của đối tượng trong game.
3. **public Transform[] cannibal\_SpawnPoints, boar\_SpawnPoint**: Hai mảng này chứa các vị trí (Transform) mà kẻ thù "cannibal" và "boar" sẽ được sinh ra (spawn) trên bản đồ.
4. **[SerializeField] private int cannibal\_Enemy\_Count, boar\_Enemy\_Count**: Hai biến này lưu số lượng kẻ thù "cannibal" và "boar" hiện tại trong game. Chúng được gán giá trị thông qua Inspector trong Unity.
5. **private int initial\_Cannibal\_Count, initial\_Boar\_Count**: Hai biến này lưu số lượng ban đầu của kẻ thù "cannibal" và "boar". Chúng có thể được dùng để khởi tạo hoặc thiết lập lại số lượng kẻ thù trong game.
6. **public float wait\_Before\_Spawn\_Enemies\_Time = 10f**: Biến này xác định khoảng thời gian chờ (tính bằng giây) trước khi sinh ra các kẻ thù mới sau mỗi đợt spawn. Nó giúp tạo ra khoảng nghỉ giữa các lần xuất hiện của kẻ thù, đảm bảo rằng trò chơi không bị quá tải với quá nhiều kẻ thù cùng một lúc.

các biến này được sử dụng để cấu hình và quản lý việc sinh ra các kẻ thù trong game, bao gồm việc xác định loại kẻ thù, số lượng, vị trí sinh ra, và thời gian chờ giữa các lần sinh ra kẻ thù.

Hàm Start trong đoạn mã này được gọi khi đối tượng chứa script được khởi tạo trong Unity

void Start()

{

initial\_Cannibal\_Count = cannibal\_Enemy\_Count;

initial\_Boar\_Count = cannibal\_Enemy\_Count;

SpawnEnemies();

StartCoroutine("CheckToSpawnEnemies");

}

**Khởi tạo số lượng ban đầu của kẻ thù**:

initial\_Cannibal\_Count = cannibal\_Enemy\_Count;

initial\_Boar\_Count = boar\_Enemy\_Count;

Dòng này gán giá trị ban đầu của số lượng kẻ thù "cannibal" và "boar" bằng với giá trị hiện tại của cannibal\_Enemy\_Count

**Gọi hàm SpawnEnemies**:

SpawnEnemies();

Dòng này gọi hàm SpawnEnemies để sinh ra các kẻ thù ban đầu trong game. Hàm này sẽ sử dụng các biến như cannibal\_SpawnPoints, boar\_SpawnPoint, cannibal\_Prefab, và boar\_Prefab để tạo ra các kẻ thù tại các vị trí chỉ định.

**Bắt đầu Coroutine CheckToSpawnEnemies**:

StartCoroutine("CheckToSpawnEnemies");

Dòng này bắt đầu một Coroutine có tên là CheckToSpawnEnemies. Coroutine là một hàm đặc biệt trong Unity, cho phép thực hiện các tác vụ theo thời gian (như chờ đợi) mà không chặn lại luồng chính của game. Coroutine này có thể kiểm tra và sinh ra thêm kẻ thù theo thời gian dựa trên các điều kiện nhất định.

Kết luận:

hàm Start thiết lập số lượng ban đầu của các kẻ thù, sinh ra các kẻ thù ban đầu, và bắt đầu một Coroutine để kiểm tra và sinh ra thêm kẻ thù khi cần thiết.

void MakeInstance()

{

if (instance == null)

{

instance=this;

}

}

Hàm MakeInstance trong đoạn mã này có nhiệm vụ đảm bảo rằng chỉ có một instance duy nhất của lớp EnemyManager tồn tại trong game, theo mẫu thiết kế Singleton.

**Kiểm tra xem instance đã được khởi tạo hay chưa**:

Dòng này kiểm tra xem biến tĩnh instance đã được gán giá trị chưa (tức là kiểm tra xem instance của EnemyManager đã tồn tại chưa).

**Gán giá trị cho instance**:

instance=this;

Nếu biến instance là null, tức là chưa có instance nào của EnemyManager tồn tại, nó sẽ gán instance bằng this. this trong ngữ cảnh này đại diện cho instance hiện tại của lớp EnemyManager mà hàm MakeInstance đang được gọi.

Điều này đảm bảo rằng chỉ có một instance duy nhất của EnemyManager trong game, và các script khác có thể truy cập vào instance này thông qua EnemyManager.instance.

MakeInstance được gọi trong Awake, đảm bảo rằng instance được thiết lập trước khi bất kỳ phương thức nào khác của EnemyManager được gọi.

void SpawnCannibal()

{

int index = 0;

for(int i=0;i<cannibal\_Enemy\_Count;i++)

{

if(index >= cannibal\_SpawnPoints.Length)

{

index = 0;

}

Instantiate(cannibal\_Prefab, cannibal\_SpawnPoints[index].position,Quaternion.identity);

index++;

}

cannibal\_Enemy\_Count = 0;

}

Hàm SpawnCannibal có nhiệm vụ sinh ra một số lượng kẻ thù "cannibal" nhất định tại các điểm spawn đã chỉ định.

**Khởi tạo biến index**:

int index = 0;

Biến index được sử dụng để chỉ định vị trí trong mảng cannibal\_SpawnPoints mà kẻ thù sẽ được sinh ra.

**Vòng lặp để sinh ra kẻ thù**:

for(int i=0;i<cannibal\_Enemy\_Count;i++)

Vòng lặp này chạy từ 0 đến cannibal\_Enemy\_Count - 1, nghĩa là nó sẽ lặp lại đúng bằng số lượng kẻ thù cần được sinh ra.

**Kiểm tra và đặt lại chỉ số index nếu cần**:

if(index >= cannibal\_SpawnPoints.Length)

{

index = 0;

}

Nếu index vượt quá hoặc bằng số lượng điểm spawn có sẵn (cannibal\_SpawnPoints.Length), nó sẽ được đặt lại về 0. Điều này đảm bảo rằng các kẻ thù được sinh ra tuần tự tại các điểm spawn, và nếu hết các điểm spawn, nó sẽ quay lại từ đầu.

Sinh ra kẻ thù "cannibal"

Instantiate(cannibal\_Prefab, cannibal\_SpawnPoints[index].position,Quaternion.identity);

index++;

Tạo ra một instance của cannibal\_Prefab tại vị trí của cannibal\_SpawnPoints[index] với không có quay (Quaternion.identity). Sau đó, tăng index để chuẩn bị cho lần lặp tiếp theo.

**Đặt lại cannibal\_Enemy\_Count về 0**:

cannibal\_Enemy\_Count = 0;

Sau khi tất cả kẻ thù đã được sinh ra, số lượng kẻ thù "cannibal" được đặt lại về 0. Điều này có thể nhằm ngăn chặn việc sinh thêm kẻ thù mà không có yêu cầu mới.

**Mục đích của hàm SpawnCannibal**

Hàm này có nhiệm vụ chính là quản lý việc sinh ra các kẻ thù "cannibal" tại các vị trí chỉ định trong game. Các kẻ thù sẽ được sinh ra theo vòng lặp và được đặt tại các điểm spawn tuần tự. Nếu hết các điểm spawn, quá trình sinh ra sẽ quay lại từ đầu danh sách điểm spawn.

Hàm SpawnBoars có nhiệm vụ sinh ra một số lượng kẻ thù "boar" nhất định tại các điểm spawn đã chỉ định.

void SpawnBoars()

{

int index = 0;

for (int i = 0; i < boar\_Enemy\_Count; i++)

{

if (index >= boar\_SpawnPoint.Length)

{

index = 0;

}

Instantiate(boar\_Prefab, boar\_SpawnPoint[index].position, Quaternion.identity);

index++;

}

boar\_Enemy\_Count = 0;

}

**Khởi tạo biến index**:

int index = 0;

Biến index được sử dụng để chỉ định vị trí trong mảng boar\_SpawnPoint mà kẻ thù sẽ được sinh ra.

**Vòng lặp để sinh ra kẻ thù**:

for (int i = 0; i < boar\_Enemy\_Count; i++)

Vòng lặp này chạy từ 0 đến boar\_Enemy\_Count - 1, nghĩa là nó sẽ lặp lại đúng bằng số lượng kẻ thù cần được sinh ra.

if (index >= boar\_SpawnPoint.Length)

{

index = 0;

}

Nếu index vượt quá hoặc bằng số lượng điểm spawn có sẵn (boar\_SpawnPoint.Length), nó sẽ được đặt lại về 0. Điều này đảm bảo rằng các kẻ thù được sinh ra tuần tự tại các điểm spawn, và nếu hết các điểm spawn, nó sẽ quay lại từ đầu.

**Sinh ra kẻ thù "boar"**:

Instantiate(boar\_Prefab, boar\_SpawnPoint[index].position, Quaternion.identity);

index++;

Tạo ra một instance của boar\_Prefab tại vị trí của boar\_SpawnPoint[index] với không có quay (Quaternion.identity). Sau đó, tăng index để chuẩn bị cho lần lặp tiếp theo.

**Đặt lại boar\_Enemy\_Count về 0**:

boar\_Enemy\_Count = 0;

Sau khi tất cả kẻ thù đã được sinh ra, số lượng kẻ thù "boar" được đặt lại về 0. Điều này có thể nhằm ngăn chặn việc sinh thêm kẻ thù mà không có yêu cầu mới.

**Mục đích của hàm SpawnBoars**

Hàm này có nhiệm vụ chính là quản lý việc sinh ra các kẻ thù "boar" tại các vị trí chỉ định trong game. Các kẻ thù sẽ được sinh ra theo vòng lặp và được đặt tại các điểm spawn tuần tự. Nếu hết các điểm spawn, quá trình sinh ra sẽ quay lại từ đầu danh sách điểm spawn.

IEnumerator CheckToSpawnEnemies()

{

yield return new WaitForSeconds(wait\_Before\_Spawn\_Enemies\_Time);

SpawnCannibal();

SpawnBoars();

StartCoroutine("CheckToSpawnEnemies");

}

Hàm CheckToSpawnEnemies là một Coroutine trong Unity, dùng để kiểm tra và sinh ra các kẻ thù (cannibals và boars) sau một khoảng thời gian chờ đợi.

**Chờ đợi một khoảng thời gian**:

yield return new WaitForSeconds(wait\_Before\_Spawn\_Enemies\_Time);

Dòng này sẽ dừng việc thực thi của Coroutine lại trong một khoảng thời gian xác định bởi wait\_Before\_Spawn\_Enemies\_Time. Thời gian chờ này có thể được cấu hình trong Unity Inspector hoặc code

**Gọi hàm SpawnCannibal**:

SpawnCannibal();

Sau khi thời gian chờ kết thúc, hàm SpawnCannibal được gọi để sinh ra các kẻ thù "cannibal" tại các vị trí spawn đã chỉ định.

**Gọi hàm SpawnBoars**:

SpawnBoars();

Sau khi sinh ra các kẻ thù "cannibal", hàm SpawnBoars được gọi để sinh ra các kẻ thù "boar" tại các vị trí spawn đã chỉ định.

**Bắt đầu lại Coroutine CheckToSpawnEnemies**:

StartCoroutine("CheckToSpawnEnemies");

Sau khi đã sinh ra các kẻ thù, Coroutine CheckToSpawnEnemies được bắt đầu lại. Điều này tạo ra một vòng lặp vô tận, nơi các kẻ thù sẽ được sinh ra định kỳ sau mỗi khoảng thời gian chờ.

**Mục đích của hàm CheckToSpawnEnemies**

Hàm này đảm bảo rằng các kẻ thù trong game được sinh ra liên tục sau mỗi khoảng thời gian định trước. Điều này giúp trò chơi luôn có các kẻ thù mới xuất hiện, tạo ra thử thách liên tục cho người chơi.

Hàm EnemyDied được thiết kế để xử lý các sự kiện khi một kẻ thù bị tiêu diệt trong trò chơi. Hàm này cập nhật số lượng kẻ thù hiện tại (cannibals hoặc boars) và đảm bảo rằng số lượng này không vượt quá số lượng ban đầu đã được thiết lập.

public void EnemyDied(bool cannibal)

{

if (cannibal)

{

cannibal\_Enemy\_Count++;

if(cannibal\_Enemy\_Count > initial\_Cannibal\_Count)

{

cannibal\_Enemy\_Count =initial\_Cannibal\_Count;

}

}

else

{

boar\_Enemy\_Count++;

if(boar\_Enemy\_Count > initial\_Cannibal\_Count)

{

boar\_Enemy\_Count=initial\_Cannibal\_Count;

}

}

}

**Mục đích và chức năng của hàm**

**Xác định loại kẻ thù bị tiêu diệt**:

Biến cannibal là một biến boolean cho biết loại kẻ thù bị tiêu diệt. Nếu cannibal là true, kẻ thù bị tiêu diệt là một "cannibal"; nếu false, kẻ thù bị tiêu diệt là một "boar".

**Cập nhật số lượng kẻ thù cannibal**:

if (cannibal)

{

cannibal\_Enemy\_Count++;

if(cannibal\_Enemy\_Count > initial\_Cannibal\_Count)

{

cannibal\_Enemy\_Count =initial\_Cannibal\_Count;

}

}

Nếu kẻ thù bị tiêu diệt là một "cannibal", biến cannibal\_Enemy\_Count được tăng thêm 1.

Nếu số lượng "cannibal" sau khi cập nhật vượt quá số lượng ban đầu (initial\_Cannibal\_Count), nó sẽ được đặt lại bằng số lượng ban đầu. Điều này ngăn không cho số lượng kẻ thù vượt quá giá trị đã định trước.

**Cập nhật số lượng kẻ thù boar**:

else

{

boar\_Enemy\_Count++;

if(boar\_Enemy\_Count > initial\_Boar\_Count)

{

boar\_Enemy\_Count=initial\_Boar\_Count;

}

}

Nếu kẻ thù bị tiêu diệt là một "boar", biến boar\_Enemy\_Count được tăng thêm 1.

Nếu số lượng "boar" sau khi cập nhật vượt quá số lượng ban đầu của cannibal (initial\_Cannibal\_Count), nó sẽ được đặt lại bằng số lượng ban đầu. Điều này dường như là một lỗi logic, vì nó nên kiểm tra với initial\_Boar\_Count thay vì initial\_Cannibal\_Count.

**Mục đích của hàm EnemyDied**

Hàm này giúp quản lý và theo dõi số lượng kẻ thù còn lại trong game, đồng thời đảm bảo rằng số lượng kẻ thù không vượt quá giới hạn ban đầu đã đặt ra. Điều này có thể quan trọng để giữ cân bằng trong game và đảm bảo rằng người chơi không phải đối mặt với số lượng kẻ thù quá lớn.

public void StopSpawning()

{

StopCoroutine("CheckToSpawnEnemies");

}

Hàm StopSpawning trong đoạn mã này có nhiệm vụ dừng việc sinh ra kẻ thù bằng cách dừng Coroutine CheckToSpawnEnemies

**Mục đích và chức năng của hàm**

StopCoroutine("CheckToSpawnEnemies");

Dòng code này sử dụng phương thức StopCoroutine để dừng Coroutine CheckToSpawnEnemies đang chạy. Coroutine này, như đã được mô tả trước đó, có nhiệm vụ sinh ra kẻ thù (cannibals và boars) sau một khoảng thời gian nhất định.

Khi nào hàm StopSpawning được sử dụng?

Hàm StopSpawning có thể được gọi trong các tình huống sau:

+Khi người chơi tạm dừng trò chơi.

+Khi trò chơi cần dừng việc sinh ra kẻ thù để thực hiện các thao tác khác như hiển thị màn hình kết thúc game, cắt cảnh, hoặc các sự kiện đặc biệt.

Như vậy, hàm StopSpawning cung cấp một cách hiệu quả để kiểm soát việc sinh ra kẻ thù trong game, giúp trò chơi trở nên linh hoạt và đáp ứng được nhiều tình huống khác nhau.