Création de niveaux supplémentaires ou gestion des checkpoints

Introduction

Les niveaux supplémentaires et la gestion des checkpoints sont des éléments clés dans de nombreux jeux vidéo. Ils offrent au joueur un sentiment de progression et permettent d'introduire de nouveaux défis. Les niveaux peuvent avoir une structure linéaire (progression d'un niveau à l'autre) ou être ouverts, avec des choix à faire pour accéder à de nouveaux défis. La gestion des checkpoints, quant à elle, permet au joueur de reprendre sa progression à partir d'un point intermédiaire en cas de défaite, augmentant ainsi la jouabilité et l'immersion.

Dans ce cours, nous allons explorer comment créer des niveaux supplémentaires dans Unity et gérer les checkpoints pour permettre aux joueurs de reprendre là où ils se sont arrêtés.

---

1. Création de Niveaux Supplémentaires

1.1 Structure de Niveaux dans Unity

Dans Unity, chaque niveau est généralement une scène distincte. Une scène représente une partie spécifique du jeu, comme un niveau ou une zone de jeu. Pour créer des niveaux supplémentaires, il vous suffit de :

1. Créer une nouvelle scène pour chaque niveau supplémentaire.
2. Ajouter des éléments spécifiques à ce niveau : nouveaux obstacles, ennemis, objets à collecter, etc.
3. Relier ces scènes entre elles via des transitions (par exemple, au moment où le joueur termine un niveau).

1.2 Transition entre les Niveaux

Les transitions entre les niveaux se font habituellement à la fin d'un niveau, lorsque le joueur atteint un certain objectif, comme la fin du niveau ou l'accomplissement d'une tâche (par exemple, tuer tous les ennemis ou collecter tous les objets). Pour cela, vous pouvez utiliser les fonctions suivantes dans Unity :

- **SceneManager.LoadScene()** : Permet de charger une scène spécifique.

Exemple de code pour changer de niveau :

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelManager : MonoBehaviour
{
    // Charge le niveau suivant
    public void LoadNextLevel()
    {
        int currentSceneIndex = SceneManager.GetActiveScene
().buildIndex;
        SceneManager.LoadScene(currentSceneIndex + 1);  //
Charge la scène suivante dans la liste des scènes
    }

    // Charge un niveau spécifique
    public void LoadLevel(int levelIndex)
    {
        SceneManager.LoadScene(levelIndex);  // Charge le n
iveau par son index
    }
}
```

Les niveaux sont ajoutés à Unity par ordre d'apparition dans la liste des scènes dans le panneau **Build Settings** (Fichier > Paramètres de construction).

1.3 Créer des Objets de Transition

Dans certains jeux, les niveaux sont reliés par des portes, des ascenseurs ou des plateformes de téléportation. Ces objets peuvent être utilisés pour initier la transition entre les niveaux.

Exemple de code pour la transition via un objet de porte :

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelTransition : MonoBehaviour
{
    public string nextLevel;  // Nom de la scène suivante

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            SceneManager.LoadScene(nextLevel);  // Charge l
e niveau suivant
        }
    }
}
```

---

2. Gestion des Checkpoints

2.1 Qu'est-ce qu'un Checkpoint ?

Un **checkpoint** dans un jeu vidéo est un point où la progression du joueur est enregistrée, permettant au joueur de revenir à ce point spécifique si le personnage meurt ou échoue, plutôt que de recommencer depuis le début du niveau. Cela est essentiel pour rendre les jeux plus accessibles et moins frustrants.

2.2 Créer des Checkpoints dans Unity

Pour créer un système de **checkpoints**, il vous faut définir des objets dans la scène qui marqueront ces points de sauvegarde. Ces objets peuvent être des zones déclencheuses où le joueur doit entrer pour enregistrer sa progression.

Étapes pour implémenter un système de checkpoints :

1. **Créer une Zone de Checkpoint :** Utilisez un `Collider` (par exemple, un `BoxCollider` ou `SphereCollider`) pour délimiter la zone du checkpoint.

2. **Enregistrer l'État du Joueur :** Lorsque le joueur atteint un checkpoint, son état (position, score, etc.) est sauvegardé.
3. **Revenir au Checkpoint :** Si le joueur meurt, il revient à ce checkpoint au lieu de recommencer le niveau depuis le début.

2.3 Exemple de Code pour un Checkpoint

```
using UnityEngine;

public class Checkpoint : MonoBehaviour
{
    public Transform respawnPoint;  // Position du checkpoi
nt
    private static Vector3 checkpointPosition;  // Dernière
position sauvegardée

    // Lorsqu'un joueur entre dans la zone de checkpoint
    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            checkpointPosition = respawnPoint.position;  //
Sauvegarde la position du checkpoint
            Debug.Log("Checkpoint atteint!");
        }
    }

    // Méthode pour réinitialiser la position du joueur au
dernier checkpoint
    public static void RespawnAtCheckpoint(Transform playe
r)
    {
        player.position = checkpointPosition;  // Réinitial
ise la position du joueur au checkpoint sauvegardé
    }
}
```

Dans ce code, le joueur entre dans un objet de type **Checkpoint**, qui sauvegarde sa position. Si le joueur meurt, il peut revenir à cette position.

2.4 Respawn et Réinitialisation des Éléments

Le respawn du joueur ne se limite pas à la position. Vous devrez peut-être également réinitialiser certains éléments du jeu (par exemple, les ennemis tués, les objets collectés, etc.). Cela peut être géré en réinitialisant l'état du jeu au moment du respawn, ce qui peut inclure la réinitialisation de la santé du joueur, des ennemis et des autres éléments pertinents.

```
public void ResetGameElements()
{
    // Exemple : Réinitialiser la santé du joueur
    playerHealth = 100;

    // Réinitialiser les ennemis
    foreach (Enemy enemy in allEnemies)
    {
        enemy.ResetPosition();  // Remet l'ennemi à sa posi
tion initiale
    }
}
```

---

3. Sauvegarde et Chargement de la Progression du Checkpoint

Dans les jeux où les checkpoints sont utilisés, il est essentiel de sauvegarder la progression du joueur. Unity offre plusieurs moyens de gérer cela :

- **PlayerPrefs** : Une option simple pour sauvegarder les données de manière clé-valeur (par exemple, la position du joueur ou le score).
- **Fichiers locaux** : Pour une plus grande flexibilité, vous pouvez utiliser des fichiers externes pour sauvegarder des données complexes comme des positions et des états du jeu.

Exemple avec PlayerPrefs pour sauvegarder la position :

```
PlayerPrefs.SetFloat("CheckpointX", checkpointPosition.x);
PlayerPrefs.SetFloat("CheckpointY", checkpointPosition.y);
```

```
PlayerPrefs.SetFloat("CheckpointZ", checkpointPosition.z);
PlayerPrefs.Save();
```

---

4. Conclusion

Les **niveaux supplémentaires** et les **checkpoints** sont essentiels pour offrir une expérience de jeu fluide et engageante. Les niveaux permettent d'introduire des nouveaux défis et des environnements variés, tandis que les checkpoints assurent que les joueurs peuvent profiter d'un défi sans devoir recommencer à zéro après chaque échec. La gestion de ces éléments dans Unity nécessite de bien comprendre les scènes, les transitions, et la sauvegarde de l'état du jeu.