Gestion des collisions et des déclencheurs (triggers)

Les collisions et les déclencheurs (ou **triggers**) sont deux des éléments les plus importants dans la gestion des interactions physiques dans les jeux vidéo. Ce cours vous expliquera comment Unity gère ces interactions à l'aide de ses composants **Collider**, **Rigidbody**, et des événements associés pour créer des mécaniques de jeu réalistes et intéressantes.

---

1. Introduction aux Collisions dans Unity

Dans Unity, les **collisions** sont des interactions physiques qui se produisent entre deux objets lorsqu'ils entrent en contact dans la scène. Ces objets doivent avoir un **Collider** attaché pour que Unity puisse détecter leur présence et la nature de l'interaction.

Composants nécessaires pour une collision :

1. **Collider** : Un Collider est un composant qui définit la forme de collision d'un objet (ex. sphère, boîte, ou forme complexe).
2. **Rigidbody** : Si vous voulez que les objets réagissent physiquement à la collision, vous devez ajouter un **Rigidbody**. Si l'un des objets a un **Rigidbody**, Unity appliquera les lois de la physique (réaction à la force, à la gravité, etc.) lors de la collision.
3. **Mode de collision** : Les collisions peuvent être gérées de manière différente selon le mode : **Collider** et **Trigger**.

Types de Colliders

- **BoxCollider** : Utilisé pour des objets à forme rectangulaire ou carrée.
- **SphereCollider** : Utilisé pour des objets sphériques.
- **CapsuleCollider** : Idéal pour des objets cylindriques comme les personnages.

- **MeshCollider** : Utilisé pour des objets avec des formes complexes (représente un maillage), mais moins performant que les autres types.

---

2. Les Types de Collisions dans Unity

Il existe deux principales catégories de collisions dans Unity :

Collision physique

Les collisions physiques impliquent des objets qui réagissent au contact selon les lois de la physique, comme les objets qui rebondissent, sont projetés, ou se déplacent lorsqu'ils entrent en collision.

Pour que cette collision physique fonctionne :

- L’un des objets impliqués doit avoir un **Rigidbody**.
- Les deux objets doivent avoir des **Colliders**.

Lorsque les objets entrent en collision, Unity génère un événement **OnCollisionEnter** et peut exécuter du code associé, comme la gestion des dégâts, des rebonds, etc.

Exemple de code :

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Obstacle"))
    {
        // Réagir à la collision, par exemple, appliquer de
s dégâts
        Debug.Log("Collision avec un obstacle !");
    }
}
```

Déclencheurs (Triggers)

Les déclencheurs, ou **triggers**, sont des zones invisibles dans le jeu qui détectent les collisions mais qui ne génèrent pas de réaction physique comme un rebond ou un arrêt. Ils sont souvent utilisés pour détecter des événements dans le jeu, comme l’entrée dans une zone, la collecte d’un objet, ou le début d’une animation.

Pour activer un **Trigger**, vous devez cocher l'option **Is Trigger** dans le **Collider**.

Quand un objet entre dans la zone d’un **Collider** en mode **Trigger**, Unity déclenche un événement appelé **OnTriggerEnter**, et vous pouvez exécuter des actions comme l'activation d’une porte, la collecte d’un objet, etc.

Exemple de code :

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        // Action à réaliser lorsqu'un objet avec le tag "P
layer" entre dans la zone du trigger
        Debug.Log("Le joueur est entré dans la zone du trig
ger !");
    }
}
```

---

3. Les Événements de Collision et de Trigger

Unity offre plusieurs événements permettant de réagir aux collisions ou déclencheurs.

Événements de Collision :

- **OnCollisionEnter(Collision collision)** : Appelé lorsque deux objets commencent à se toucher.
- **OnCollisionStay(Collision collision)** : Appelé chaque fois qu'un objet continue à entrer en contact avec un autre objet.
- **OnCollisionExit(Collision collision)** : Appelé lorsque deux objets cessent d'entrer en contact.

Exemple d’utilisation :

```
void OnCollisionEnter(Collision collision)
{
    // Code à exécuter lorsque l'objet entre en collision
    Debug.Log("Collision commencée avec " + collision.gameO
bject.name);
}
```

```
void OnCollisionStay(Collision collision)
{
    // Code à exécuter tant que l'objet est en contact avec
l'autre
    Debug.Log("Collision continue avec " + collision.gameOb
ject.name);
}

void OnCollisionExit(Collision collision)
{
    // Code à exécuter lorsque l'objet cesse d'être en cont
act
    Debug.Log("Fin de la collision avec " + collision.gameO
bject.name);
}
```

Événements de Trigger :

- **OnTriggerEnter(Collider other)** : Appelé lorsque le Collider du joueur (ou d'un autre objet) entre dans un **Trigger**.
- **OnTriggerStay(Collider other)** : Appelé chaque fois qu'un objet reste dans la zone du **Trigger**.
- **OnTriggerExit(Collider other)** : Appelé lorsque l'objet quitte la zone du **Trigger**.

Exemple d'utilisation :

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        // Code pour exécuter lors de l'entrée du joueur da
ns la zone du trigger
        Debug.Log("Le joueur entre dans la zone de déclench
eur !");
    }
}
```

```
void OnTriggerStay(Collider other)
{
    if (other.CompareTag("Player"))
    {
        // Code à exécuter tant que le joueur reste dans la
zone
        Debug.Log("Le joueur est toujours dans la zone du d
éclencheur");
    }
}

void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Player"))
    {
        // Code à exécuter lorsque le joueur quitte la zone
        Debug.Log("Le joueur quitte la zone de déclencheu
r");
    }
}
```

4. Différences entre Collision et Trigger

Aspect	**Collision**	**Trigger**
Comportement	Réaction physique (rebond, arrêt, déplacement)	Aucune réaction physique, seulement la détection
Propriétés du Collider	Pas de modification, peut être attaché à un **Rigidbody**	Le Collider doit avoir l'option **Is Trigger** activée
Événements associés	**OnCollisionEnter**, **OnCollisionStay**, **OnCollisionExit**	**OnTriggerEnter**, **OnTriggerStay**, **OnTriggerExit**
Utilisation	Détecter des chocs physiques ou des interactions physiques (ex. impacts, sauts)	Détecter des événements dans la scène sans effets physiques (ex. collecte d'objets, entrée dans une zone)

5. Cas d'Utilisation des Triggers

Les **Triggers** sont utilisés dans de nombreux cas dans les jeux vidéo :

- **Collecte d'objets** : Par exemple, lorsque le joueur entre dans la zone d'un objet à ramasser, comme une clé ou un power-up.
- **Activation de mécanismes** : Lorsqu'un personnage entre dans une zone pour déclencher un événement, comme l'ouverture d'une porte ou le lancement d'une animation.
- **Déclenchement d'une zone dangereuse** : Par exemple, entrer dans une zone qui inflige des dégâts au joueur (comme un piège).

---

6. Optimisation des Collisions et Triggers

L'utilisation excessive des collisions et des triggers peut entraîner une baisse des performances, surtout dans les scènes avec de nombreux objets.

Voici quelques bonnes pratiques pour optimiser les collisions et les triggers dans Unity :

1. **Utiliser des colliders simples** : Préférez des **BoxCollider**, **SphereCollider** et **CapsuleCollider** plutôt que des **MeshCollider**, qui sont plus gourmands en ressources.
2. **Désactiver les collisions non nécessaires** : Si un objet n'a pas besoin de réagir physiquement, vous pouvez désactiver les collisions ou les triggers de manière dynamique pour éviter les calculs inutiles.
3. **Utiliser des couches de collision** : Utilisez les **Layers** pour spécifier avec quels objets un Collider ou un Trigger doit interagir.

Conclusion

La gestion des collisions et des déclencheurs est un aspect fondamental de la physique des jeux vidéo. Comprendre comment utiliser efficacement les **Colliders** et **Triggers** vous permet de créer des interactions dynamiques et intéressantes dans vos jeux, tout en optimisant les performances. En combinant les collisions physiques pour les réactions réalistes et les déclencheurs pour les événements non physiques, vous pouvez concevoir des mécaniques de jeu complexes et immersives.