



# R507 Dispositifs interactifs

Project Object Collector



## Objectifs

- Apprendre à travailler avec des assets
- comprendre les canvas et l'UI

## Leçons

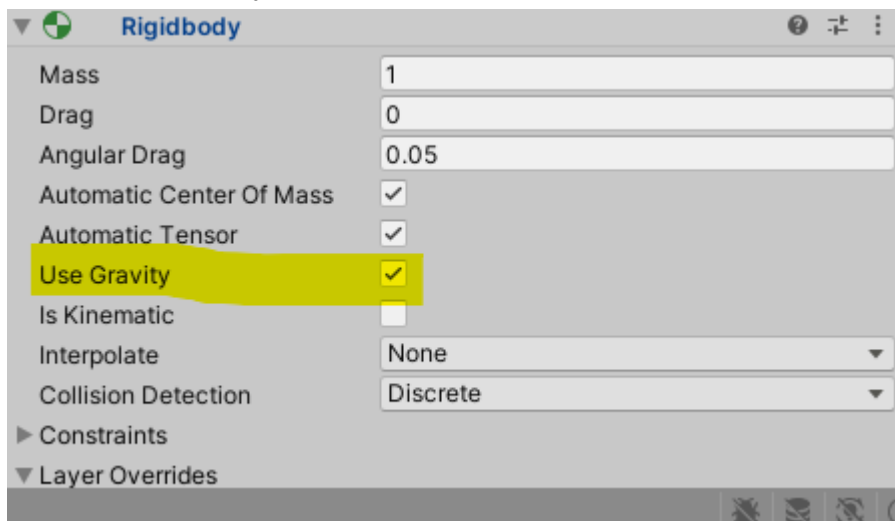
- Physics
- Gravity
- Forces
- Collision (using OnCollisionEnter)
- Raycast
- Canvas
- UI

# 1- Gravity

La gravité dans Unity est un vecteur qui représente la force de gravité appliquée à tous les objets rigides dans la scène. La gravité est définie comme un vecteur de 3 dimensions, où la direction vers le bas est négative. Par défaut, la gravité est définie à 9,81 m/s<sup>2</sup> dans la direction négative de l'axe Y, ce qui correspond à la gravité terrestre.

La gravité peut être modifiée:

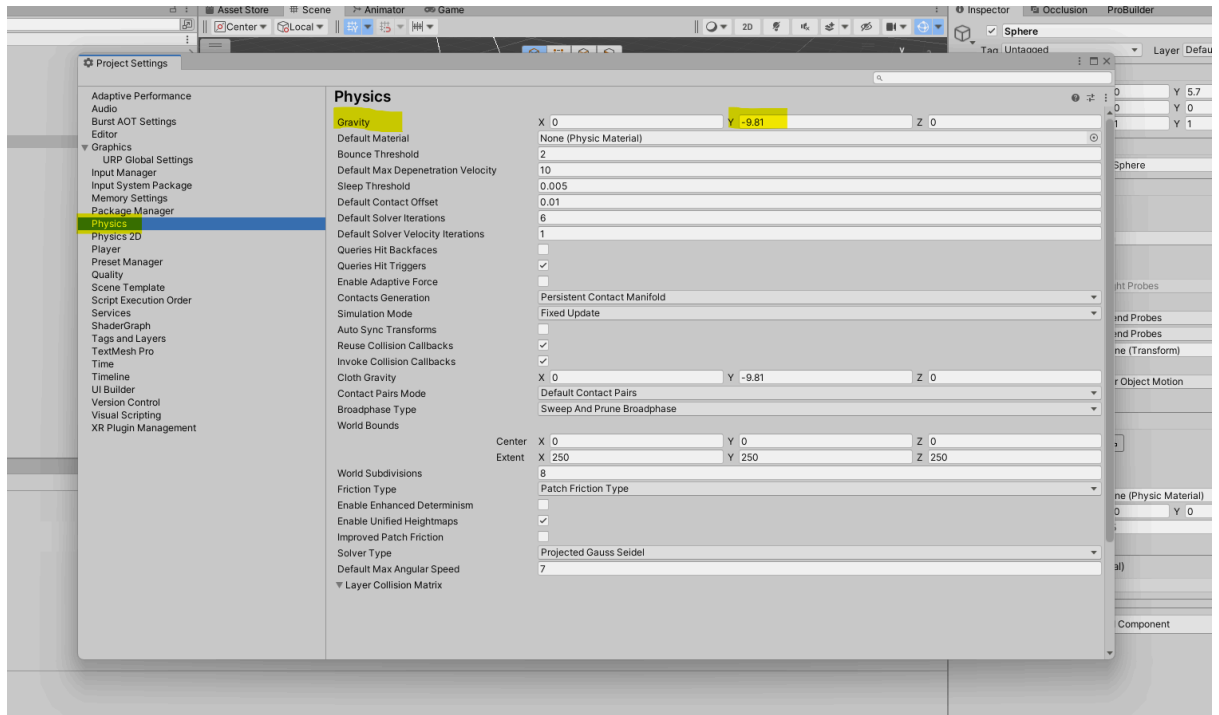
- pour un objet spécifique en utilisant le composant Rigidbody et en définissant la propriété useGravity sur false.



- pour la gravité globale de la scène en modifiant la propriété Physics.gravity dans un script.

```
Physics.gravity = new Vector3(0,-5,0);
```

- Pour l'ensemble de votre projet dans les settings.



## Démonstration

Modification de la gravité

C'est à vous :

1. Créez un nouveau script Gravity.cs
2. Déclarez deux variables public float gravityX et gravityY
3. Dans le Start de la classe modifiez la gravité  

```
Physics.gravity = new Vector3(gravityX, gravityY, 0);
```
4. Dans votre scène ajouter un objet de type cube
5. Associez lui un rigidbody
6. Associez lui le nouveau script Gravity.cs
7. Modifiez les valeur de gravityX et gravity Y
8. Cliquez sur play

## 2- Force

La force dans Unity est une quantité vectorielle qui peut être appliquée à un objet pour le déplacer ou le faire tourner. Les forces peuvent être utilisées pour simuler des interactions physiques telles que la gravité, la friction, la poussée, etc.

Dans Unity, vous pouvez appliquer une force à un objet en utilisant la fonction `AddForce` du composant `Rigidbody`. Cette fonction prend un vecteur de force en trois dimensions et un mode de force en option.

Les modes de force disponibles sont `Force`, `Acceleration`, `Impulse` et `VelocityChange`.

## Force

Permet d'ajouter une force de manière continue au rigidbody en tenant compte de sa masse.

## Acceleration

Permet d'ajouter une force de manière continue au rigidbody en ignorant sa masse. Comme si la mass était à 1.

## Impulse

Donne une impulsion de manière ponctuelle en prenant en compte la masse

## VelocityChange

Donne une vélocité de manière ponctuelle en ignorant la masse

ForceMode			
Know the Difference			
for rigidBody.AddForce(f, ForceMode)*:			
	time dependent	time independent	
mass dependent	<b>Force</b> same as: $v += f * dt / m$	<b>Impulse</b> same as: $v += f / m$	Use these when force depends on mass. E.g. the same amount of gunpowder fires a small bullet at a high velocity, and a big one at a low velocity.
mass independent	<b>Acceleration</b> same as: $v += f * dt$	<b>VelocityChange</b> same as: $v += f$	Use these when the mass doesn't affect the change of velocity. An example is gravity, which accelerates all objects equally.
Use these when you want to apply a force continuously, each frame, like a rocket engine, jetpack, magnet..		Use these when you want to apply an instantaneous force, such as on collision reponse, shooting a bullet, kicking a ball..	
* The same rule applies to all related functions that use ForceMode as a parameter			

plus d'information : [Unity - Scripting API: ForceMode \(unity3d.com\)](https://docs.unity3d.com/Scripting/ForceMode.html)

## Démonstration

Ajout de force avec différents forceMode

### C'est à vous :

1. Ajouter 4 cube à votre scène.
2. Pour chacun d'eux ajouter un rigidbody (avec gravity disable) et un nouveau script AddForceScript.cs
3. Dans le Script:

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  Script Unity | 0 références
6  public class AddForceScript : MonoBehaviour
7  {
8      // Start is called before the first frame update
9      public ForceMode forceMode;
10     public Vector3 forceToAdd = Vector3.zero;
11
12     Message Unity | 0 références
13     void Start()
14     {
15         GetComponent<Rigidbody>().AddForce(forceToAdd, forceMode);
16     }
17 }
```

4. Modifiez les paramètres pour chaque cube : un forceMode différent sur chaque cube, une force identique sur chaque cube
5. Cliquez sur play

## 3- Collision

La collision dans Unity est un système qui permet de détecter les interactions entre les objets dans un environnement 3D. Les collisions sont détectées lorsque les colliders de deux objets se touchent ou se chevauchent. Les colliders sont des composants qui définissent la forme d'un objet et sont utilisés pour détecter les collisions avec d'autres objets. Les colliders peuvent être des formes simples telles que des sphères, des boîtes ou des capsules, ou des formes plus complexes telles que des maillages.

Lorsqu'une collision est détectée, Unity envoie un message à chaque objet impliqué dans la collision. Vous pouvez utiliser ces messages pour déclencher des événements tels que des effets sonores, des animations ou des points de score. Vous pouvez également utiliser les messages pour modifier le comportement des objets, par exemple en faisant rebondir une balle lorsqu'elle touche un mur.

Pour détecter les collisions, vous devez ajouter un composant Rigidbody à au moins un des objets impliqués dans la collision. Le Rigidbody ajoute de la physique à l'objet, comme la gravité, la masse et la vitesse. Vous devez également ajouter un composant Collider à chaque objet impliqué dans la collision. Les colliders définissent la forme de l'objet et sont utilisés pour détecter les collisions avec d'autres objets.

OnTriggerEnter et OnCollisionEnter sont deux fonctions de rappel dans Unity qui sont appelées lorsqu'un objet entre en collision avec un autre objet. La principale différence entre les deux est que OnTriggerEnter est appelé lorsque deux objets avec des colliders configurés comme "Trigger" entrent en collision, tandis que OnCollisionEnter est appelé lorsque deux objets avec des colliders configurés comme "Solid" entrent en collision 12.

Lorsque OnTriggerEnter est appelé, les objets ne sont pas physiquement affectés par la collision. Au lieu de cela, un message est envoyé aux scripts attachés aux objets pour leur permettre de réagir à la collision. Par exemple, vous pouvez utiliser OnTriggerEnter pour détecter quand un joueur entre dans une zone spécifique de la scène.

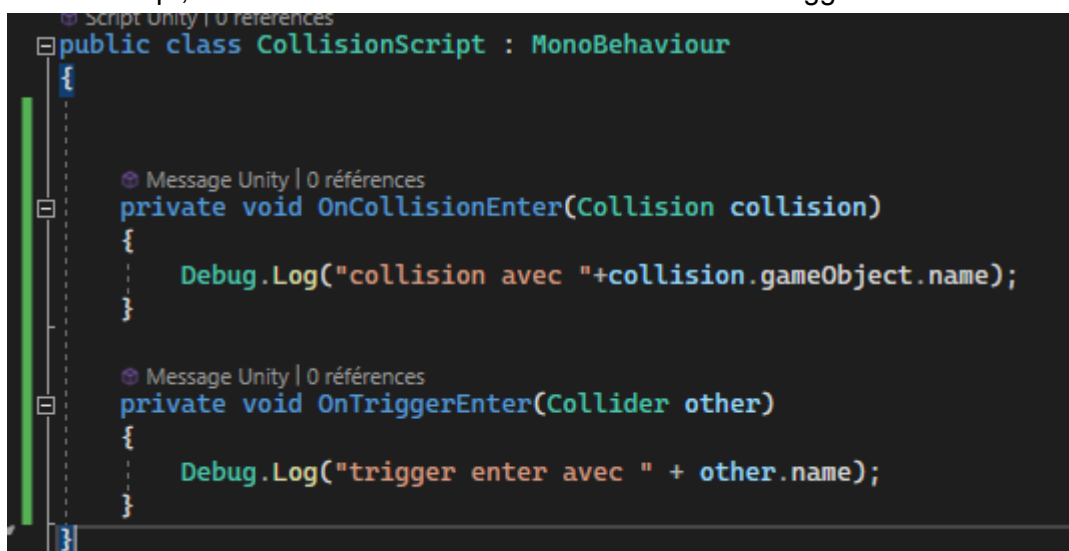
Lorsque OnCollisionEnter est appelé, les objets sont physiquement affectés par la collision. Par exemple, si un joueur entre en collision avec un mur, il sera repoussé par le mur. Vous pouvez utiliser OnCollisionEnter pour détecter quand un joueur entre en collision avec un ennemi ou un objet destructible.

## Démonstration

Collision avec onTrigger dans le script puis avec on Collision

### C'est à vous :

1. Créez un nouveau script CollisionScript.cs
2. Associez le à vos cubes
3. Dans le script, utilisez les méthodes OnCollisionEnter et OnTriggerEnter.



```
Script Unity | 0 références
public class CollisionScript : MonoBehaviour
{
    Message Unity | 0 références
    private void OnCollisionEnter(Collision collision)
    {
        Debug.Log("collision avec "+collision.gameObject.name);
    }
    Message Unity | 0 références
    private void OnTriggerEnter(Collider other)
    {
        Debug.Log("trigger enter avec " + other.name);
    }
}
```

4. Cliquez sur play et faites bouger un des cubes jusqu'à la collision, un log collision apparaît.

5. Sélectionnez tous les cubes et passez le paramètre isTrigger du rigidbody à true.
6. refaite une collision, le log est maintenant un trigger enter

## 4- UI & Canvas

L'interface utilisateur (UI) dans Unity est un système qui permet de créer des éléments d'interface utilisateur pour les jeux et les applications. Les éléments d'interface utilisateur peuvent inclure des boutons, des barres de progression, des menus déroulants, des champs de texte, etc. L'interface utilisateur est utilisée pour permettre aux utilisateurs d'interagir avec le jeu ou l'application.

Unity fournit un ensemble de composants d'interface utilisateur qui peuvent être utilisés pour créer des éléments d'interface utilisateur. Les composants d'interface utilisateur incluent Button, Text, Image, Slider, Toggle, InputField, etc. Vous pouvez également créer des éléments d'interface utilisateur personnalisés en utilisant des scripts C#.

Pour créer une interface utilisateur dans Unity, vous devez ajouter un Canvas à votre scène. Le Canvas est un conteneur pour les éléments d'interface utilisateur. Vous pouvez ensuite ajouter des éléments d'interface utilisateur au Canvas en faisant glisser et en déposant des composants d'interface utilisateur à partir de la fenêtre Project sur le Canvas.

Une fois que vous avez créé vos éléments d'interface utilisateur, vous pouvez les personnaliser en modifiant leurs propriétés dans l'inspecteur. Vous pouvez également ajouter des événements à vos éléments d'interface utilisateur en utilisant des scripts C#. Par exemple, vous pouvez ajouter un événement OnClick à un bouton pour déclencher une action lorsque le bouton est cliqué.

### **Démonstration**

créer une interface de type formulaire ou paramètre de jeux

## 5- Project Object Collector

Le but de ce projet est de créer un jeu. L'objectif de ce jeu est de cliquer sur les items qui apparaissent à l'écran un peu à la manière de "fruit ninja". Certains items font gagner des points, d'autres font perdre des points.

## Création du projet

1. Dans le unity HUB, créez un nouveau projet qui se nommera "project \_Object Collector" avec le template "3D core"
2. Récupérez le zip AssetObjectCollector.zip sur moodle
3. Dézippez le fichier
4. Importez les assets dans unity, depuis le menu principal , sélectionnez Assets > Import Package > Custom Package  
sélectionnez Prototype-5\_Starter-Files.unitypackage file
5. Dans la fenêtre d'explorateur de projet, ouvrez la scène Prototype5.
6. Dans la fenêtre de visualisation de la scène, cliquez sur le bouton 2D afin de visualiser la scène sur un plan en 2 dimensions.

## Changement de la texture de fond

1. Vous pouvez dès à présent modifier l'image de fond et les images de côté pour personnaliser votre jeu.

## Création des cibles

1. Depuis le dossier "Course Library" faites un Drag and drop de 3 "bons" objets et d'un objet "malus" vers la scène.
2. Ajoutez un Rigidbody et un collider à chacun de ces objets.
3. Vérifiez que les collider sont bien ajustés à chaque objet.
4. Créez un nouveau dossier Scripts dans lequel vous créez un nouveau script Target.cs.
5. Attachez ce script à chacun des objets que vous avez mis dans la scène.
6. Créez un nouveau dossier Prefabs puis Glissez/déposez chacun des 4 objets dans le dossier afin d'en faire des prefabs.
7. Supprimez les objets de la scène.

## Lancement des cibles

1. Ouvrez le Script Target.cs
2. Écrivez le Script de telle manière à ce que les objets target soient balancés vers le haut
3. Les points d'origine du lancé doivent variés à chaque lancé et de manière aléatoire

## Création d'un GameManager

1. Dans votre scène, créez un objet vide que vous appellerez GameManager
2. Ajoutez lui un nouveau Script GameManager.cs
3. Faites en sorte que ce script instantie aléatoirement des objets (prefabs que vous avez préalablement réalisés)
4. pour lancer de manière régulière des objets vous pourrez utiliser la notion de coroutine avec IEnumerator



**Définition : IEnumerator**

IEnumerator est une interface de C# qui permet de définir une méthode qui peut être suspendue et reprise. Dans Unity, 'IEnumerator' est souvent utilisé pour créer des coroutines. Les coroutines sont des fonctions qui peuvent être exécutées en parallèle avec d'autres fonctions, ce qui permet de créer des effets de délai, des animations et des mouvements de manière plus facile et plus efficace. Exemple:

```
IEnumerator ExampleCoroutine()  
{  
    Debug.Log("Coroutine started");  
    yield return new WaitForSeconds(1);  
    Debug.Log("Coroutine ended");  
}
```

## Destruction des objets suite à un clic.

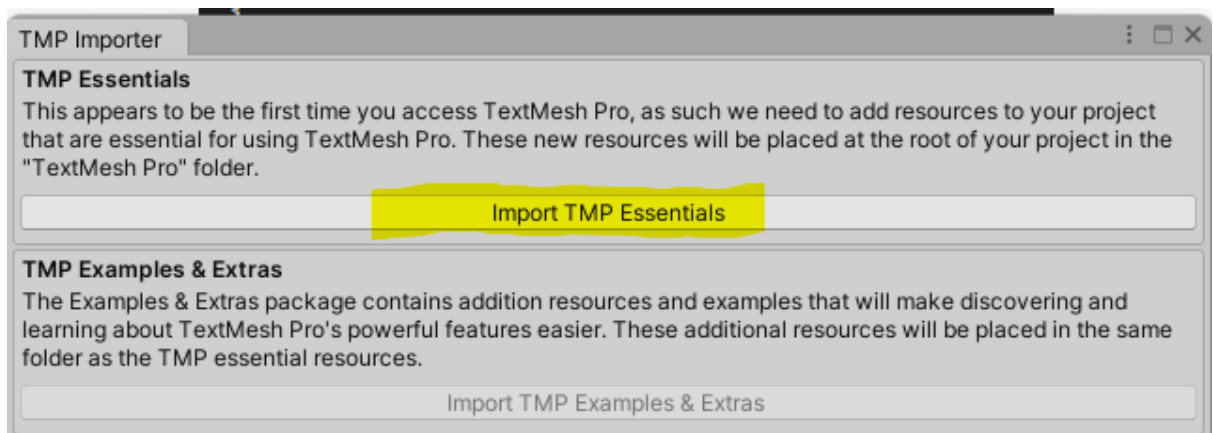
1. Dans le script Target.cs
2. Ajoutez une méthode OnMouseDown(). Dans cette méthode, détruire le GameObject

## Destruction des objets en collision avec un trigger.

1. Faites en sorte que les target soit détruite lorsqu'elles entrent en contact avec l'objet Sensor

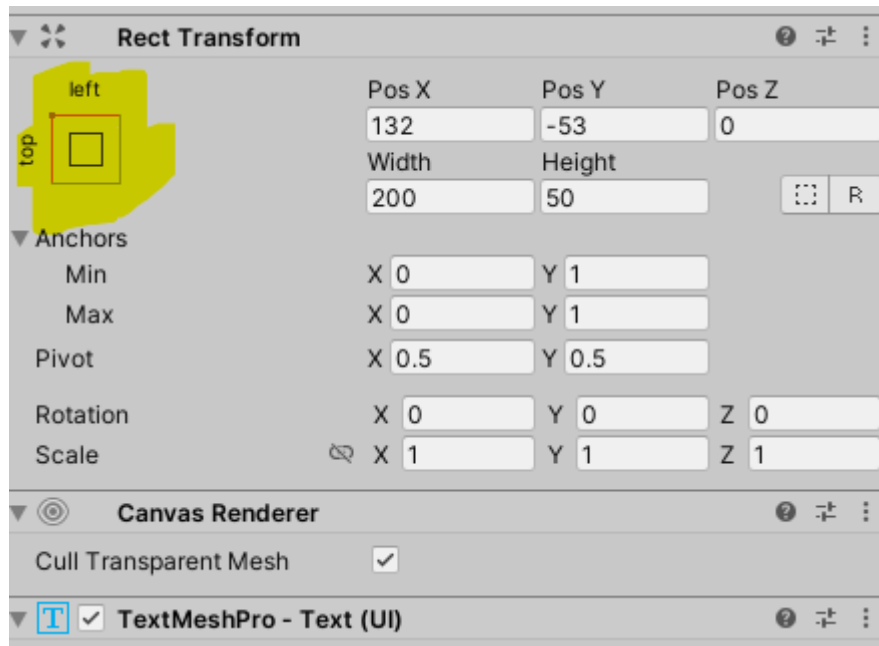
## Ajout d'un score

1. Dans la hiérarchie de la scene, faire un clic droit et créer un nouvel élément UI>TextMeshPro
2. si une popup apparaît, cliquez sur "import TMP Essentials"



3. Renommer l'objet "ScoreText"

4. Dans le component transform modifiez son “anchor”



5. Puis modifiez sa position de telle sorte à le placer dans le coin haut /gauche de l'écran. Pour ajuster la position vous pouvez afficher simultanément le visuel du jeu et celui de la scène.
6. modifiez le texte en “score”
7. modifiez ensuite la font le style , la taille et la couleur pour que cela corresponde au style de votre jeu.

## Initialisation du score

1. Dans le GameManager, ajoutez l'utilisation du namespace TMPro.

```
using TMPro;
```

2. Dans le GameManager toujours, créez une nouvelle variable public de type TextMeshProUGUI et associez la variable dans l'éditeur
3. Dans le GameManager, faire en sorte d'initier le score à 0 et d'afficher le score à l'écran avec un rendu du type “score : 0 ”

## Mise à jour du score

1. Dans le GameManager, ajoutez une méthode UpdateScore qui permettra de mettre à jour l'affichage du score.
2. Dans la méthode que vous venez de créer, mettre à jour la variable score avant de mettre à jour l'affichage.
3. Dans le start, remplacer la modification de votre score par un appel à la nouvelle méthode.
4. Dans la méthode SpawnTarget, faites un appel à la méthode UpdateScore qui ajoutera 5 points à chaque fois qu'un nouvel objet “spawn”.

## Ajouter des points à la destruction d'enemy

1. Dans la classe target.cs, donnez une valeur de point au target et faire en sorte que lorsqu'il est détruit le joueur gagne des points.
2. Dans les prefabs que vous avez créé, initiez une valeur positive pour les bonus et négative pour le malus.

## Ajouter des particules destruction d'enemy

1. Dans la classe target.cs, ajoutez une variable public de type ParticleSystem au nom de explosionParticle
2. Dans l'inspecteur, associez des prefab particle que vous trouverez dans le dossier Course Library> Particles.
3. Puis, faites en sorte d'instancier des particules dans le onMouseDown

## Ajouter d'un text Game Over

1. Faites un clic droit sur le canvas , ajoutez un nouveau TextMeshPro que vous nommerez "GameOverText"
2. Configurez le dans l'inspecteur : position, text, font, size, style, color et alignement
3. Désactiver ce gameObject en décochant la checkBox active

## Affichage du Game Over

1. Dans le GameManager.cs, ajoutez une variable public TextMeshProUGUI au nom de gameOverText et assignez lui (via l'inspecteur) le texte que vous venez de créer.
2. Faire en sorte que si un objet bonus n'est pas touché par le joueur et touche le Sensor alors le texte Game over s'affiche à l'écran (pensez à créer une méthode GameOver() dans le game Manager)

## Gestion du GameOver

1. Dans le GameManager, ajoutez une variable public de type bool isActive;
2. Dans le Start de la méthode, passez isActive à true, et dans la méthode GameOver, passez la variable à false.
3. Faire en sorte que les objets bonus/malus soient instanciés si le jeu est actif

## Ajouter un bouton "Restart"

1. Faites un clic droit sur le canvas pour y associer un nouveau bouton.
2. Nommez ce bouton "restartButton"
3. Positionnez correctement le bouton
4. Sélectionnez le text du bouton et personnalisez le : color, font, size etc ...

## Recommencer le jeu au clic sur le bouton “Restart”

1. Dans le GameManager, ajoutez le classname SceneManagement
2. Créez une nouvelle méthode RestartGame qui va recharger la scene courante.
3. Dans l'inspector du bouton restart. ajouter un nouvel appel à méthode dans le onclick qui fera un appel à la méthode RestartGame

## Affichage du bouton “Restart”

1. Via l'inspector, désactiver par défaut le bouton restart
2. Dans le GameManager, ajoutez une variable public de type bouton à laquelle vous pouvez associer votre bouton restart.
3. Dans la méthode GameOver activer le bouton

## Ajout d'un titre et de la gestion de la difficulté

1. Dupliquer le text GameOver et renommer le pour en faire un titre
2. Dupliquer le bouton restart et modifiez le pour en faire un bouton “facile”
3. Dupliquer le bouton “facile” à deux reprises pour en faire deux autres boutons “normal” et “difficile”
4. Pour ces trois nouveaux boutons supprimer l'appel à la méthode restartGame dans le OnClick
5. Créez un nouveau Script DifficultyButton.cs et attachez le aux 3 boutons.
6. faire en sorte que lors d'un click sur un de ces bouton, le jeu commence. Pour cela vous devrez aussi créer une méthode StartGame qui initialise le jeu dans le GameManager.

## Faire disparaître l'écran titre au début du jeu

1. Dans le canvas, ajoutez un objet vide que vous nommerez “titleScreen”.
2. Faire un drag and drop du titre et des trois boutons dans cet objet vide
3. Dans le GameManager ajoutez une nouvelle variable public de type GameObject que vous pouvez appeler titleScreen. Associez votre titleScreen à cette variable dans l'inspector.
4. Dans la méthode StartGame du GameManager désactiver l'objet TitleScreen

## Passage de paramètre de difficulté

1. Dans la classe DifficultyButton, ajoutez une variable difficulty de type float que vous pouvez initier à 1 pour easy, 2 pour medium et 3 pour hard
2. Faire en sorte que cette variable définisse la difficulté du jeu.

# Intégration de kinect à votre projet pour remplacer le controller.

Pour utiliser kinect sur un PC il est nécessaire de télécharger le SDK et le package unity dispo à cette adresse:

<https://learn.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows#tools-and-extensions>

La vidéo ci-dessous vous explique comment procéder:

<https://www.youtube.com/watch?v=m3BAIJAGIkQ&t=2s>

Pour utiliser le kinect comme controller, les 3 premières vidéos de ce tutoriel devraient vous aider.

[https://www.youtube.com/watch?v=hKDaI\\_E7rDg](https://www.youtube.com/watch?v=hKDaI_E7rDg)