Date : 16/10/2023

Séance 7 – Relations entre classes et objets sur Unity

Objectifs :

- Introduire les notions des relations entre classes
- Mise en pratique de concepts décrits sur le moteur de jeu Unity

Concepts :

Relations d'association : les relations d'association modélisent le passage d'information (ou communication) entre les objets des différentes classes. Une association peut être entre deux classes ou entre plusieurs. Sans savoir, vous avez déjà utilisé des relations dans nos projets de discipline.

Par exemple, dans plusieurs occasions nous avons utilisé un script (qui devient un objet lorsqu'il est attaché à un GameObject) pour modifier les attributs d'un objet « Transform », responsable pour le déplacement d'un « GameObject ».

Nous avons aussi utilisé des relations pour passer des informations entre un objet script qui contrôlait le déplacement d'une voiture vers un objet qui géré la quantité d'essence restante, la classe « *GestionVoiture* ». Ce dernier objet renvoyait en retour si la voiture pouvait continuer à rouler.

Nous nous intéressons maintenant à deux types particuliers d'association : **l'agrégation** et la **composition**. Ceux deux types de relations modélisent la relation objet composite /composant (ou tout/partie). La différence pratique entre elles peut être faite par rapport au cycle de vie des objets modélisés. Si l'objet composant n'existe pas sans l'objet composite, c'est une relation de composition. Néanmoins, si le composant continuer à exister après la destruction de l'objet composite, la relation est une agrégation.

Par exemple, les objets qui modélisent les roues d'un objet « voiture » n'ont pas de raison d'exister lorsque l'objet voiture est détruit. Donc, une composition. Imaginons maintenant le cas d'un jeu de course avec plusieurs pistes et plusieurs voitures. Une piste aura des relations d'agrégation avec plusieurs voitures, vu qu'à la fin de chaque course, les voitures pourront passer à une autre piste, mais la piste cessera d'exister.

Exemple d'une relation de composition

```
public class Voiture : MonoBehaviour
{
    public GameObject roueAvantDroite   = null;
    public GameObject roueAvantGauche   = null;
    public GameObject roueArriereDroite = null;
    public GameObject roueArriereGauche = null;
```

Nous pourrions aussi utiliser le code source au-dessus pour coder une relation d'agrégation. C'est le projet du jeu qui dira quel type d'association nous utiliserons et la façon que nous allons signaler la destruction des objets composés qui fera la différence entre ces deux types de relations dans le code source.

Navigation des objets composites : lorsque vous ajoutez un GameObject 3D dans la scène (un cube, une sphère ou un objet plus avancé) vous pouvez regarder sur la fenêtre « Inspector » qu'il contient plusieurs composants par défaut, tels que : un MeshRenderer, un Transform et un BoxCollider. Nous avons couramment augmenté cette liste avec les scripts, par exemple.

La bibliothèque de classes Unity nous propose plusieurs méthodes pour nous aider à naviguer les objets dans un GameObject avec le langage C#, tels que : *GetComponent*, *GetComponenetInChildren* and *Find*.

Méthode « GetComponent » : elle nous permet d'accéder à un composant d'un objet « GameObject ». Par exemple, le MeshRenderer, le Transform et le BoxCollider. L'objet sera cherché par rapport au type saisi entre les symboles « < » et « > ».

Exemple :

```
Renderer apparence = getComponent<Renderer>();
```

Méthode GetComponentInChildren : cette méthode renvoie la liste des composants du type spécifié qui appartient au GameObject cherché et à ses sous-objets. Le code ci-dessous initialise une liste avec les composants « Transform » de tous les sous-objets contenu dans « gameObject », son « Transform » inclus.

Exemple :

```
Transform[] liste2 = gameObject.GetComponentsInChildren<Transform>();
```

La boucle **foreach** est bien adaptée pour itérer sur une liste d'éléments. Par exemple :

```
foreach (Transform transf in liste2)
{
  Debug.Log("Trnsf:" + transf.name + "," + transf.GetType());
}
```

GameObject.Find : cette méthode cherche un objet « GameObject » dans la scène par son nom. Le code ci-dessous cherche dans la scène un objet appelé « MaCamera ».

```
GameObject lien = GameObject.Find("MaCamera");
```

Pour tester si un objet a été trouvé, vous pouvez simplement utiliser la variable qui recevra le résultat de la recherche comme condition dans un test conditionnel.

```
GameObject lien = GameObject.Find("MaCamera");
if (lien){
     Debug.Log("L'objet a été trouvé");
}
else{
     Debug.Log("L'objet n'a pas été trouvé");
}
```

Exercice 1

Créez un nouveau projet de jeu 3D sur Unity et réalisez les actions suivantes :

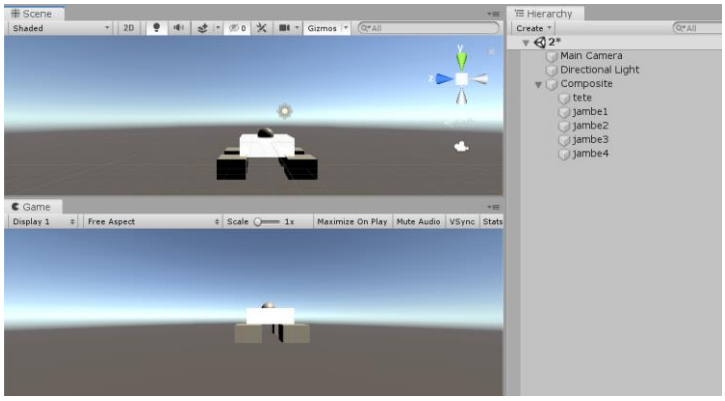
1. **Accès à un composant**. Ajoutez un GameObject 3D à votre scène. Ajoutez un script à votre objet qui cherche le composant « MeshRenderer » du GameObject et change sa couleur lorsqu'une touche du clavier est appuyée. Vous pouvez utiliser la classe « Color »[1] pour choisir la couleur à affecter à l'objet.

[1] https://docs.unity3d.com/ScriptReference/Color.html

Accès à l'attribut couleur du composant « MeshRenderer » :

```
Renderer -> material -> color
```

2. **Objet composite**. Créez un objet 3D composé d'autres objets 3Ds comme l'exemple ci-dessous. Nommez chaque sous-objet différemment. Utilisez les méthodes de navigation vues pour afficher les noms de chaque sous-objets d'un objet composite sur la console. Attention, le script doit être attaché à l'objet composite. Ensuite, faites que la taille des sous-objets « jambes » soit réduite de 25 % à chaque fois que nous appuyons sur la touche « R ». Vous devez chercher les sous-objets cible par nom.



Exercice 2 - Création d'un jeu de type « Coureur » en perspective 2,5D

Import et placement des élément dans la scène

1. Téléchargez le prototype de jeu 3 proposé par Unity : « Prototype 3 - Runner »
2. Créez un nouveau projet de jeu de type « 3D » et importez le « prototype 3 » dans votre projet.

   Vous pouvez supprimer la scène rajoutée par défaut à votre projet afin de garder seulement la scène incluse dans le projet par le prototype.

3. Vous pouvez changer l'arrière-plan de votre jeu en modifient le champ « Sprite » dans le composant « Sprite Renderer » de l'objet « Background ».
4. Choisissez un personnage (« Characters ») et placez-le dans la scène. Ajoutez les composants « RigidBody » et « BoxCollider » au personnage.
5. Ajoutez un script à votre personnage. Ce script comportera les fonctions nécessaires pour lui contrôler. Nous l'appellerons « Controle Personnage ».

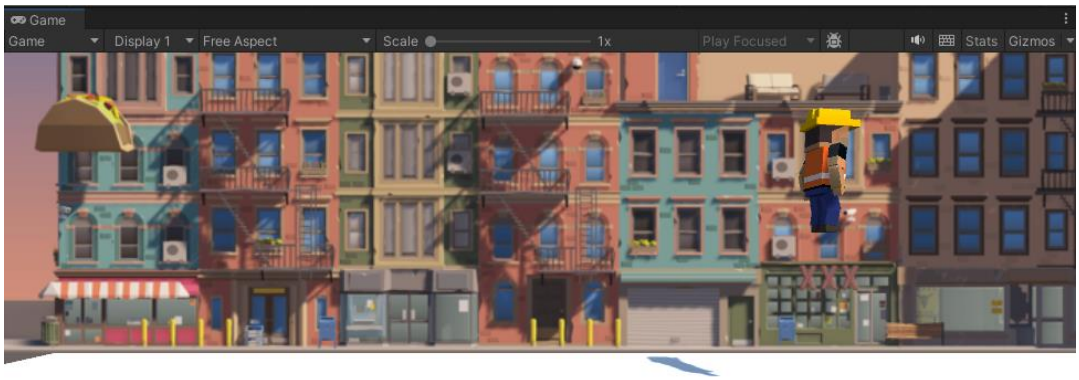
Contrôle du personnage

Nous allons maintenant ajouter la fonctionnalité de « sauter » au personnage. Rajoutez les éléments suivants au script « Controle Personnage ».

6. Créez un attribut afin de stocker une référence au composant « RigidBody » du personnage. Par exemple, « instanceRigidBody ».
7. Utilisez la méthode « GetComponent » dans la méthode « Start » du script afin d'initialiser l'attribut « instanceRigidBody » avec une référence à l'objet « RigidBody » du personnage.
8. Dans la méthode « Update », utilisez la méthode « AddForce » du composant « RigidBody » afin de faire le personnage sauter lorsque qu'on appuie sur la touche « espace » du clavier.

```
//L'exemple de code ci-dessous applique à l'objet une force de « 50 » unités vers le bas de la scène
instanceRigidBody .AddForce(Vector3.down*50, ForceMode.Impulse) ;
```

- Choisissez les valeurs adaptées pour la direction et l'intensité de la force, ainsi que le type de force.
- Pensez à utiliser un attribut pour stocker la valeur d'intensité de la force à être appliquée à votre personnage afin de permettre de tester différents valeurs à partir de l'onglet « Inspector ».



9. Modifiez le paramètre « Physics.gravity » dans la méthode « Start » du script « Controle Personnage »afin de modifier l'effet de la gravité sur le saut du personnage.

```
//le code ci-dessous augment la force de la gravité de 3 fois.
Physics.gravity *= 3 ;
```

Pensez à ajouter un attribut à votre script afin de stocker le facteur de modification de la gravité que vous voulez appliquer à votre jeu.

10. Choisissez des valeurs pour l'attribut d'intensité de force de saut et le modificateur de gravité afin d'obtenir un saut avec un comportement naturel.

Dans l'état actuel du jeu, votre personnage peut sauter indéfiniment, et ainsi monter dans la scène sans restriction. Nous allons maintenant utiliser la détection de collisions afin d'éviter ce comportement.

11. Créez un attribut booléen dans le script afin de savoir si le personnage touche le sol.

L'attribut prend la valeur de vrai si le personnage touche le sol, et faux sinon.

12. Choisissez la méthode de collision adaptée afin d'identifier lorsque le personnage rentre en collision avec le sol.
13. Ajoutez une contrainte au code source qui gère le saut du personnage afin d'éviter que le personnage puisse sauter s'il ne touche pas le sol.

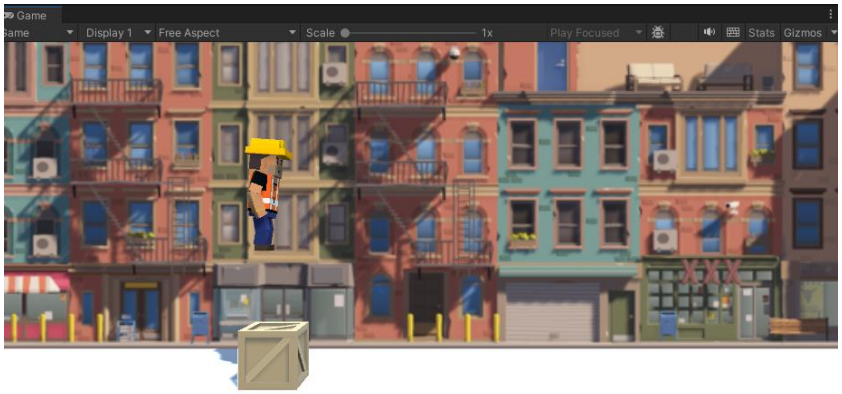
Création d'un modèle d'obstacle

Maintenant que le personnage est capable de sauter dans le jeu, nous allons créer des obstacles afin de rendre son parcours plus amusant.

14. Ajoutez un objet du dossier « /Course Library/Obstacles/ » au jeu.
15. Rajoutez des composants « RigidBody » et « Collider » à l'objet choisi.
16. Rajoutez un script à l'objet qui lui déplace vers la gauche.

La vitesse de déplacement de l'objet doit être paramétrable via la fenêtre « Inspector »

17. Testez votre objet « obstacle » dans la scène. Lorsque l'objet est fonctionnel, créez un modèle « prefab » à partir de cet objet.

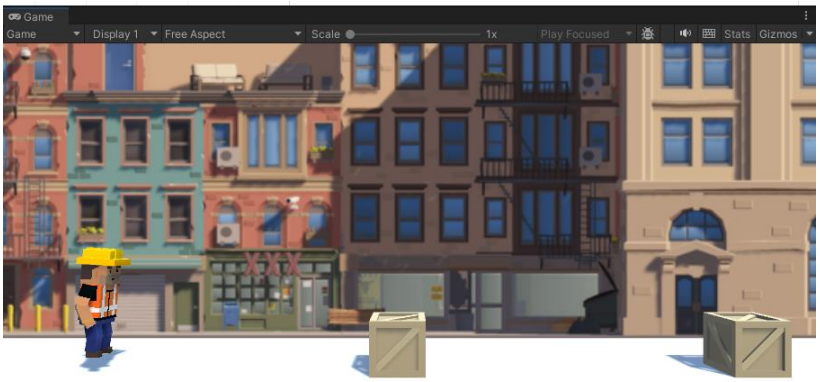


Création automatique d'obstacles

Nous disposons maintenant d'un modèle d'obstacle pour notre jeu. Notre prochain objectif est de créer automatiquement des obstacles dans la scène à intervalles réguliers.

1. Ajoutez un objet de type « Empty GameObject » à votre scène pour vous en servir en tant que gestionnaire du jeu. Ajoutez un script à cet objet afin de contrôler la création des obstacles dans la scène.
2. Utilisez les méthodes « Instantiate » et « InvokeRepeating » vues pendant la séance précédente afin de créer des obstacles dans la scène en utilisant le modèle d'obstacle créé.

- Le modèle d'obstacle à utiliser doit être modifiable via la fenêtre « Inspector ».
- Différemment de la séance précédente, la position de création des obstacles est fixe.
- Utilisez un attribut afin de définir la position initiale des obstacles.
- N'oubliez pas que la méthode « InvokeRepeating » n'accepte pas d'appeler une fonction qui requiert d'arguments.



Animation de l'arrière-plan et gestion du jeu

Pendant cet exercice, nous allons travailler sur l'animation de l'arrière-plan du jeu.

1. Ajoutez à l'arrière-plan du jeu (objet « background ») une copie du script créé précédemment qui déplace un objet à gauche.

Votre arrière-plan se déplace maintenant en créant une animation, mais il arrive éventuellement au bout de la scène.

2. Ajoutez un deuxième script à l'objet « background », nommé « RepetitionArrierePlan ». Ce script sera responsable pour remettre l'objet « background » à sa position au démarrage du jeu à chaque fois que l'arrière-plan du jeu sort du cadre de la scène.

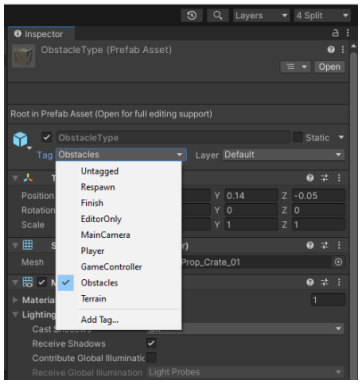
> Vous pouvez rajouter un attribut « RepetitionArrierePlan » afin de stocker la position de l'arrière-plan au démarrage du jeu.

Condition de fin du jeu

Maintenant que nous avons ajouté des comportements au personnage, aux obstacles et à l'arrière-plan, nous allons travailler sur la détermination de la fin du jeu et son effet sur les éléments du jeu.

3. Ajoutez une étiquette (*tag*) nommée « Obstacle » au modèle « prefab » des obstacles afin de pouvoir les identifier par un identifiant unique. Vous pouvez faire de même pour le terrain du jeu (étiquette « Terrain) et le personnage (étiquette « Player »).

> Pour rajouter une étiquette à un objet, sélectionnez l'objet (ou « prefab ») d'intérêt et utilisez l'option « Add tag » sur le champ « Tag », accessible via la fenêtre « Inspector ».



Script « Contrôle Personnage »

4. Ajoutez un attribut de type booléen au script qui contrôle le personnage. Cet attribut aura pour objectif d'identifier l'état du jeu (faux : le jeu est en cours, vrai : le jeu est fini).
5. Modifiez le code source du script qui contrôle le personnage afin qu'il puisse prendre en compte des différents types de collision, en utilisant le système d'étiquettes mis en place.

> - Si le personnage fait une collision avec le sol, il est à nouveau capable de sauter.
> - S'il fait une collision avec un obstacle, le message « Fin du jeu » est affiché à l'aide de la méthode « Debug.Log() ». Pensez à modifier l'attribut qui indique la fin du jeu.

Scripts « Gestion Jeu » et « DeplacerGauche »

6. Faites le nécessaire afin d'arrêter la création d'objets ainsi que le déplacement de l'arrière-plan lorsque le jeu est fini.

> - Vous pouvez créer un attribut du même type que le script qui contrôle le personnage (e.g., « ControlePersonnage ») dans les scripts qui gère la création d'objets et le déplacement des objets à gauche.
> - La méthode GameObject.Find (« nom de l'objet ») peut vous aider à obtenir une référence à l'objet « personnage ». En utilisant cette référence, vous pouvez obtenir une référence vers le script de gestion du personnage avec la méthode « GetComponent<TYPEOBJET>() », et ainsi suivre la valeur de l'attribut « fin du jeu ».
> - Vous pouvez utiliser la méthode « CancelInvoke(« nom de la méthode ») » pour arrêter la création d'obstacles.

Script « DeplacerGauche »

Vous avez probablement remarqué que les obstacles créés pendant le jeu s'accumulent dans votre scène. Vous pouvez les visualiser en regardant la fenêtre « Hierarchy ».

7. Modifiez le script « Déplacer à gauche » afin qu'il détruise les objets de type « obstacle » qui sortent du cadre de la scène.

- Pensez à utiliser le système de « tag » afin de détruire seulement les objets « obstacle ».
- Utilisez un attribut de votre script afin de définir à partir de quelle position (ou dimension), vous considérez qu'un objet est à l'extérieur de la scène et doit être ainsi détruit.

Considérations finales

Une partie des exercices de ce TD a été adaptée de l'unité 3 du Parcours Unity Learn - Junior Programmer. N'hésitez pas à visiter leur formation en ligne afin d'avoir accès au cours complet, ainsi qu'à avancer en autonomie sur les sujets que n'ont pas été traité en cours.

Références

- https://docs.unity3d.com/ScriptReference/GameObject.GetComponent.html
- https://learn.unity.com/tutorial/lesson-3-1-jump-force?uv=2021.3&pathwayId=5f7e17e1edbc2a5ec21a20af&missionId=5f7648a4edbc2a5578eb67df&projectId=5cf9639bedbc2a2b1fe1e848#5ce35aa5edbc2a29e31b3c71