Date : 17/09/2024

Séance 4 – Calcul vectoriel

Précédemment dans les séances 2 et 3 :

Pendant les séances précédentes vous étiez amené à coder le déplacement d'un objet sur Unity. Votre script doit probablement ressembler au code source ci-dessous. Le code ci-dessous déplace l'objet d'intérêt suivant l'axe Z avec une vitesse de 4.5 mètres. La direction de déplacement est définie par un vecteur de type « Vector3 ». Dans notre exemple, nous avons utilisé la méthode « Time.deltaTime » pour considérer le temps passé entre l'appel précédent et l'appel actuel à la méthode « Update » dans le calcul de la nouvelle position de l'objet d'intérêt.

```
public class deplacement : MonoBehaviour {
 public float vitesse = 4.5f;
 void Update(){
      gameObject.transform.position += new Vector3(0,0,1)*vitesse*Time.deltaTime;
 }
}
```

Objectifs :

- Rappel des opérations sur des vecteurs
- Introduction aux classes Vector2 et Vector3
- Mise en pratique des concepts appris

Concepts :

Le moteur de jeu Unity utilise des vecteurs pour réaliser différentes tâches liées à la mise en place et le déroulement d'un jeu vidéo, tels que le calcul de la position des objets ou de la direction et de la vitesse de leur mouvement. La Figure 1 montre les repères Unity pour des jeux en 2D et 3D.

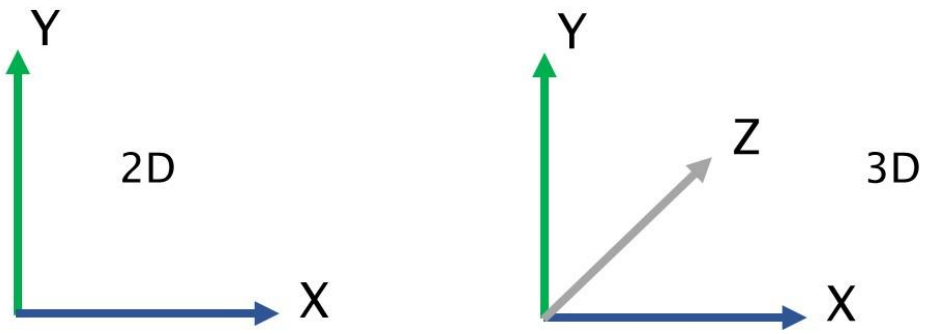

Figure 1. Repères Unity en 2D (gauche) et 3D (droite)

Classes « Vector2 » et « Vector3 » : la bibliothèque Unity met à notre disposition des classes pour calculer la majorité des opérations liées à la manipulation des vecteurs. Comme ces noms le suggèrent, la classe *Vector2* est utilisée pour représenter des vecteurs en 2 dimensions, < x, y >, et la classe *Vector3* pour des vecteurs en 3D dimensions, < x, y, z >.

Pour créer un vecteur sur Unity, vous devez utiliser l'opérateur « **new** ». L'exemple ci-dessous montre la création des variables pour stocker les vecteurs < 3,5 > et < 2,-1 >.

```
Vector2 v2D_a = new Vector2 (3, 5); // vecteur 2D
Vector2 v2D_b = new Vector2 (2, -1); // vecteur 2D
```

Pour extraire les coordonnées d'un vecteur, nous utilisons les attributs « x » et « y » pour un vecteur 2D et « x », « y » et « z » pour les vecteurs 3D.

```
Debug.Log(v2D_a.x);
Debug.Log(v2D_a.y);
```

L'addition entre deux vecteurs crée un nouveau vecteur où chaque dimension est la somme des dimensions correspondent des vecteurs sommés. Considérons l'exemple suivant :

```
a = <3,5>
b = <2,-1>
c = a + b
c = <3+2, 5+(-1)>
c = <5, 4> //résultat
```

La Figure 2 illustre l'addition des vecteurs « a » et « b », en noir, et leur résultat en gris, le vecteur « c ».

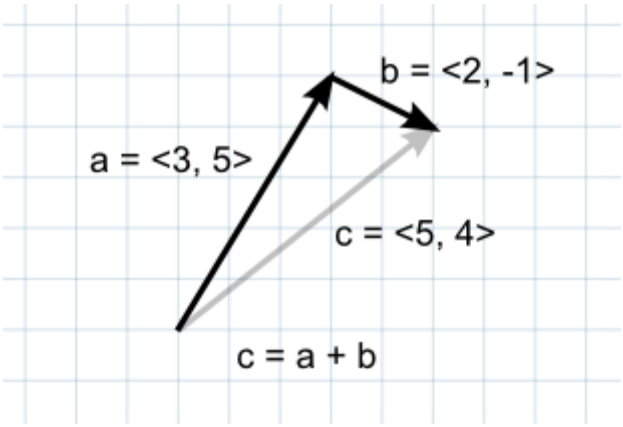

Figure 2. Sommes de vecteurs. Source : Unity.com

Les opérations d'addition et de soustraction entre deux variables de type vecteur sur Unity sont faites avec les opérateurs « + » et « - ». Vous trouverez ci-dessous un exemple d'addition.

```
Vector2 v2D_c;
v2D_c = v2D_a + v2D_b;
Debug.Log("Résultat: " + v2D_c);
//ou
Debug.Log("Résultat (v2).x : " + v2D_c.x +", .y :"+ v2D_c.y)
```

Déplacement des objets : il y a plusieurs façons de déplacer un objet sur Unity. Précédemment, nous avons directement modifié l'attribut « position » de l'objet. Une alternative, c'est d'utiliser la méthode « Transform.Translate() ». L'exemple ci-dessous produit un déplacement équivalant à celui produit par la solution précédente, mais en utilisant la méthode « Translate() ».

Exemple d'utilisation de la méthode « Translate »

```
transform.Translate ( new Vector3(0,0,1) * Time.deltaTime * vitesse );
```

Jusqu'à présent, nous avons créé un vecteur pour représenter la direction que nous voulions donner au mouvement de notre objet. Néanmoins, les classes de type vecteur sur Unity proposent plusieurs raccourcis pour faciliter le déplacement des objets dans la scène par

rapport au repère du monde. Le Tableau 1 montre quelques raccourcis proposés par la classe « Vector3 » :

Tableau 1. Raccourcis de direction proposés par la classe « Vector3 »

Raccourci	Équivalent à (x, y, z)
Vector3.back	Vector3(0, 0, -1).
Vector3.down	Vector3(0, -1, 0).
Vector3.forward	Vector3(0, 0, 1).
Vector3.left	Vector3(-1, 0, 0).
Vector3.right	Vector3(1, 0, 0).
Vector3.up	Vector3(0, 1, 0).
Vector3.zero	Vector3(0, 0, 0).
Vector3.one	Vector3(1, 1, 1).

Nous pouvons donc réécrire les exemples précédents de déplacement avec le raccourci « Vector3.forward » pour déplacer l'objet d'intérêt vers l'avant.

Exemple 1

```
transform.Translate ( Vector3.forward * Time.deltaTime * vitesse );
```

Exemple 2

```
gameObject.transform.position+= Vector3.forward * vitesse * Time.deltaTime;
```

De façon similaire, les raccourcis « Vector3.up », « Vector3.down », « Vector3.left » et « Vector3.right » définissent le déplacement de l'objet vers le haut, vers le bas, vers la gauche et vers la droite de la scène, respectivement.

Rotation des objets : nous pouvons également faire tourner un objet autour d'un axe à l'aide de la méthode « Transform.Rotate() ». Cette méthode prend comme arguments l'axe à utiliser pour la rotation (sous la forme d'un vecteur) et la vitesse de rotation.

```
transform.Rotate (Vector3.up, vitesseRotation * Time.deltaTime);
```

Repère de l'objet : les raccourcis présentés ci-dessus se basent sur le repère du monde. Mais, il est parfois nécessaire d'utiliser le repère de l'objet. Par exemple, lorsque nous voulons faire en sorte que l'objet se déplace en fonction de son orientation. Autrement dit, qu'il avance vers la direction qu'il pointe. Dans ce cas, nous devons utiliser les raccourcis proposés par le composant « Transform » de l'objet d'intérêt : « *.forward* », « *.right* » et « *.up* ».

Le moteur de jeu Unity propose aussi des méthodes pour nous aider à réaliser des fonctionnalités courantes d'un jeu vidéo. Dans ce cadre, vous pouvez utiliser deux méthodes importantes telles que « MoveTowards » et « LookAt ».

Méthode « Vector3.MoveTowards » : cette méthode calcule le vecteur pour déplacer un objet A vers un objet B. Son résultat peut être ensuite utilisé pour calculer la nouvelle position de l'objet à déplacer.

Exemple d'utilisation de la méthode « MoveTowards(Vector3,Vector3, float) » :

```
Vector3.MoveTowards(objetDInteret.transform.position,  gameObject.transform.position, 20);
```

Méthode « LookAt ». La classe « Transform » met à notre disposition la méthode « LookAt » qui tourne l'objet courant vers l'objet d'intérêt. Pour l'utiliser, nous devons appeler la méthode « LookAt » à partir du composant « transform » de l'objet qui observe. Cette méthode prend comme argument le composant « transform » de l'objet à regarder.

Exemple d'utilisation de la méthode « Transform.LookAt » :

```
void update(){
      //transform : composant « Transform » de l'objet qui       regarde
      //objet_d_interet : instance de type GameObject de l'objet à regarder
      transform.LookAt(objet_d_interet.transform);
}
```

Exercices

Calcul des vecteurs

1. Calculez à la main les opérations entre vecteurs ci-dessous :
   a) Addition entre vecteurs 2D :

```
Position actuelle : <+5,+8>
Vitesse : <+3,+2>
```

   b) Soustraction des vecteurs en 2D :

```
Position : <-1,-3>
Déplacement : <-2,+2>
```

   c) Addition entre vecteurs 3D

```
Position actuelle : <-2,-1,+5>
Vitesse : <+1,+4,+3>
```

   d) Soustraction des vecteurs en 3D :

```
Position : <2,-4,+1>
Déplacement : <-1,-1,+3>
```

Vecteurs sur Unity

2. Créez un nouveau projet 3D sur Unity. Créez un script comportemental et utilisez les classes « Vector2 » et « Vector3 » pour coder les opérations entre vecteurs ci-dessus et valider vos résultats. Les résultats des opérations doivent être stockés eux aussi sous la forme de variables de type vecteur. Votre code doit être placé dans la méthode « Start ». Créez une instance de GameObject « Empty » pour attacher votre script et le tester.
3. Ensuite, ajoutez un « GameObject » de type « Cube » sur votre scène et placez-le sur la position <0,0,0>. Modifiez sa taille (scale) pour l'allonger selon l'axe Z, par exemple (1, 1, 10). Faites-le avancer de 3 unités par seconde. N'oubliez pas de la méthode « Time.deltaTime » sur le calcul du déplacement de votre jeu. Testez votre jeu !
4. Ensuite, sélectionnez la caméra du jeu (objet « Maint Camera ») et paramétrez-la avec les valeurs ci-dessous. Vous devez retrouver un résultat similaire à la Figure 3.
   - Position : <0,50,0>
   - Rotation : <90,0,0>

- Projection : Perspective
- Field of View : 120 degrés

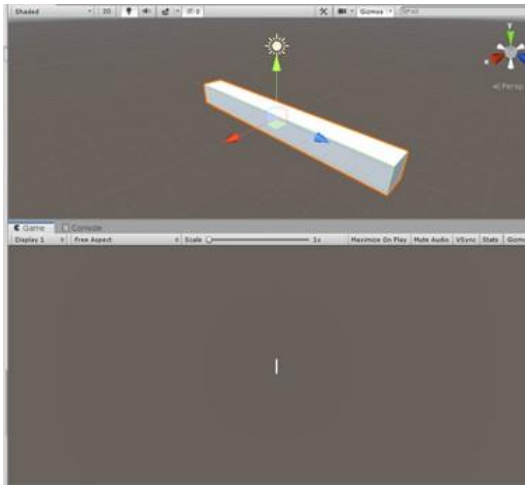

Figure 3. Résultat attendu

5. Commentez votre code pour déplacer l'objet. Maintenant, faites tourner votre objet autour de lui même de 45 dégrées à chaque « frame ». Testez votre code !
6. Finalement, combinez les deux fonctionnalités précédentes (Translate et Rotate) pour faire l'objet tourner en cercle autour de la scène.

Projet voiture

Reprenez votre projet Unity des séances 3 et 4 pour les exercices à suivre. Assurez-vous que la voiture se déplace automatiquement à chaque « frame ». La vitesse doit être un attribut du script qui déplace la voiture.

7. **Placement de la caméra :** sélectionnez la caméra du jeu (« main camera ») et placez-la à côté de la voiture du joueur. Utilisez les outils « Move tool » et « Rotate tool » pour obtenir un résultat qui ressemble à l'image ci-dessous.

Figure 4. Outils pour la manipulation des GameObject

Figure 5. Position de la caméra

8. **Regarder l'objet d'intérêt** : dans certains jeux, il est souvent nécessaire que la caméra suive le personnage du jeu. Pour accomplir cette tâche, ajoutez un script à caméra du jeu, nommez-le « regard.cs ». Ensuite, créez un attribut, « objet_cible » dans ce script pour stocker une référence vers le « GameObject » à suivre par la caméra. Utilisez les méthodes vues pour tourner la caméra vers la voiture dès que la voiture se déplace (voir Figure 6).

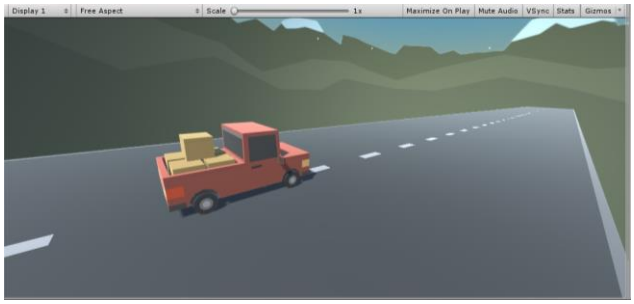

Figure 6. Retournement de la caméra

Pour lier l'objet « voiture » au script « regard.cs », cliquez sur l'objet « camera » de votre projet (sous le volet « Hierarchy »), puis déplacez l'objet « voiture » (aussi sous le volet « Hierarchy ») vers l'attribut « objet cible » du script, qui est visible sous le volet « Inspector ».

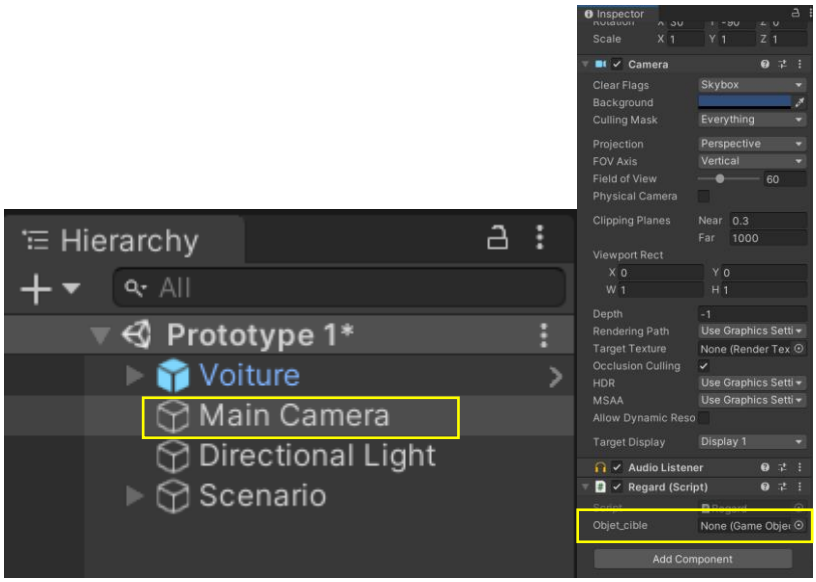

Figure 7. Volets « Hierarchy » et « Inspector »

9. **Une caméra qui accompagne le véhicule**. Faites que la caméra du jeu se déplace avec la voiture une fois que la voiture s'éloigne trop de son champ de vue. Utilisez les méthodes vues dans cette séance et l'attribut « objet_cible » pour accomplir cet exercice. L'attribut « objet_cible » doit être paramétrable via l'Inspector.

   Testez votre code avec la voiture du joueur.

Défis : faites que la caméra se déplace avec la voiture mais continue de lui être parallèle.

10. **Contrôle de la voiture** : ajoutez un script, « Deplacement.cs », pour contrôler le mouvement de la voiture. Les contrôles doivent permettre d'accélérer ou décélérer la voiture (0 à 50 m/s, méthode « Translate ») ou de la tourner vers la gauche ou la droite

(méthode « Rotate »). Vous pouvez utiliser le code source ci-dessous pour capturer l'interaction du joueur avec les flèches du clavier.

```
if    (Input.GetKeyUp(KeyCode.LeftArrow))    {…}
else if (Input.GetKeyUp (KeyCode.RightArrow)){…}
else if (Input.GetKeyUp (KeyCode.UpArrow))    {…}
else if (Input.GetKeyUp (KeyCode.DownArrow)) {…}
```

Perfectionnement :

11. **Des roues qui roulent :** Pour l'instant, la voiture contrôlée se déplace, mais ses roues ne bougent pas. Écrivez un script pour faire tourner une roue. Ensuite, appliquez ce script à l'une des roues de la voiture pour tester votre code. Une fois réussi, appliquez le script aux autres roues. N'oubliez pas de modérer la vitesse des rotations par les temps passés à travers les appels à la méthode « Update ».
12. **Vitesse partagée** : Faites en sorte que lorsque nous accélérons/décèlerons la voiture, les roues adoptent une vitesse proportionnelle à celle de la voiture.
13. **Défis II** : Faites en sorte que les roues avant tournent une fois que nous tournons la voiture vers la gauche ou la droite.