Atelier 7 – TP2

Instanciation dynamique et simulation physique

420-SF3-RE
Développement de programmes dans un environnement graphique

Auteur : Vincent Echelard

Table des matières

A. Mise en contexte

Ce travail vous permettra principalement d'intégrer les éléments des derniers ateliers portant notamment, mais pas exclusivement, sur l'instanciation dynamique et sur les moteurs physiques. Nous toucherons donc à quelques éléments clés des différentes thématiques explorées jusqu'à présent dans ce cours, soit :

- les mouvements dans l'espace 3D;
- l'instanciation dynamique ; et
- les collisions.

B. Description sommaire du travail pratique

Votre nouvel employeur, le studio VBG[1], ayant été particulièrement impressionné par votre dernier logiciel (un cube et des sphères qui faisaient des mouvements complexes dans l'espace) vous demande de participer au développement d'un prototype de leur nouveau jeu d'aventure spatial.

Ultimement, dans ce jeu, le joueur devra contrôler un vaisseau spatial qui se déplacera dans un champ de débris stellaire en combattant des extraterrestres sanguinaires. Pour votre prototype, vous devrez simplement créer un labyrinthe de caisses et le parcourir en tirant des cibles. Il faut bien commencer quelque part…

Voici les spécifications auxquelles devra répondre votre prototype :

1. La construction du niveau se fera de façon dynamique. Vous devrez analyser un fichier texte contenant une description symbolique du niveau. C'est-à-dire, la position des caisses et des cibles, ainsi que la position de départ du vaisseau et celle du point de sortie.

2. Les déplacements du vaisseau ainsi que le tir des projectiles se feront de manière « physique », c'est-à-dire en appliquant des forces. Vous ne devrez jamais manipuler directement le composant ***Transform*** du vaisseau ou des projectiles.

[1] Very Boring Games

C. Description des tâches à réaliser

1. Préambule

Le directeur technique du studio a déjà conçu l'architecture du prototype et il a déjà commencé la codification d'une portion des scripts de ce prototype. Cela va diminuer votre charge de travail, mais va vous forcer à respecter un certain nombre de lignes directrices en ce qui concerne le nom des scripts et leur interface publique de programmation, que ce soit le nom des attributs modifiables dans l'éditeur ou les méthodes publiques des différents scripts. Votre projet comportera ultimement les scripts suivants :

1. GameObject Camera
    a. Script DéplacementCaméra
2. Prefab Cible
    a. Script GestionCible
    b. Script GestionSectionCible
3. GameObject GameManager
    a. Script Carte
    b. Script FsmJeu
4. GameObject Pointage
    a. Script GestionPointage
5. Prefab Projectile
    a. Script ActionProjectile
6. Prefab Vaisseau
    a. Script ActionVaisseau
    b. Script ComportementVaisseau
    c. Script InputManager
    d. Script MouvementVaisseau
7. Prefab Mine
    a. Script ComportementMine (bonus)
8. Prefab Destination
    a. Script ComportementAtome (bonus)
    b. Script ComportementÉlectron (bonus)

2. Script DeplacementCamera (à compléter)

Ce script est un composant du ***GameObject CameraPrincipale***. Il permet de faire en sorte que la caméra reste positionnée au-dessus du vaisseau lors des déplacements de ce dernier. La caméra n'est toutefois pas affectée par les rotations effectuées par le vaisseau. Ce script est en grande partie implémentée, il ne reste que quelques lignes de code à rédiger.

3. Script GestionCible (à compléter)

Ce script est un composant du ***Prefab Cible***. Il permet de gérer les trois sections de la cible. Il contient trois méthodes :

- La méthode **`Update()`** qui permet de faire pivoter la cible sur elle-même *(à rédiger)*.
- La méthode **`InitialiserComportementCible()`** qui permet d'initialiser les différentes sections de la cible *(déjà rédigée)*.
- La méthode **`DétruireCible()`** qui doit être appelée lors d'une collision d'un projectile avec une des sections de la cible *(déjà rédigée)*.

4. Script GestionSectionCible (à compléter)

Ce script est un composant de chacune des sous-sections du ***Prefab Cible***. Il permet de gérer la collision d'un projectile avec la section de la cible porteuse dont ce script est un composant. Il contient au minimum deux méthodes :

- La méthode **`InitialiserSectionCible()`** qui permet de conserver un lien vers le script qui permet de gérer le panneau de pointage *(à rédiger)*.
- La méthode **`OnCollisionEnter()`** qui permet de réagir à une collision impliquant la section de la cible *(à rédiger)*. Dans le cas d'une collision avec un projectile, ce script devra faire modifier le pointage affiché en ajoutant la valeur associée à la section de la cible touchée et devra provoquer la destruction de la cible. Il n'y a rien de particulier à faire ici dans le cas d'une collision avec un autre objet de la scène tel que le vaisseau par exemple.

5. Classe Carte (déjà rédigée)

Cette classe n'est pas un script de comportement et n'est donc pas directement associée à un ***GameObject***. Il s'agit d'une classe « classique » qui permet de lire et d'analyser le fichier texte contenant la description du niveau. Cette classe est rédigée, mais vous devrez la lire et la comprendre pour être capable de l'utiliser correctement. Vous pourrez constater qu'étant donné qu'il ne s'agit pas d'un script de comportement, cette classe est instanciée à l'aide de l'instruction ***new***.

6. Script FsmJeu (à compléter)

Ce script est un composant du ***GameObject GameManager***. Il s'agit du script s'occupant de la gestion générale des différentes étapes du jeu. Dans le cas présent, cette gestion se fait au travers d'une machine à états finis (*finite state machine* ou FSM). Dans ce script, la structure générale de la machine à états finis est déjà rédigée, mais il reste à rédiger quelques méthodes qui sont appelées par la machine à états finis.

7. Script GestionPointage (à compléter)

Ce script est un composant du ***GameObject Pointage*** qui est situé dans la hiérarchie de la caméra. Mis à part la méthode d'initialisation, les autres méthodes ce script sont appelées lors des collisions des différentes collisions gérées par les autres scripts du jeu, pour modifier le pointage en conséquence.

8. Script ActionProjectile (à compléter)

Ce script est un composant du ***Prefab Projectile***. Il permet de gérer l'autodestruction du projectile, ainsi que la collision du projectile avec un autre objet de la scène. Il contient au minimum trois méthodes :

- La méthode **`Update()`** qui permet de gérer l'autodestruction du projectile une fois que la durée de vie du projectile est écoulée *(à rédiger)*.
- La méthode **`OnCollisionEnter()`** qui permet de détruire le projectile lorsqu'il entre en contact avec un autre objet de la scène, excluant le vaisseau naturellement *(à rédiger)*.
- La méthode **`InscrireProjectile()`** qui est appelée par le script ***ActionVaisseau*** lors de l'instanciation du projectile et qui permet d'établir un lien entre le vaisseau et le projectile pour faciliter la gestion du nombre de projectiles actifs *(à rédiger)*.

Note	*La destruction du projectile s'accompagne, dans tous les cas, d'une explosion (voir le **prefab Explosion**).*

9. Script ActionVaisseau (à compléter)

Ce script est un composant du ***Prefab Vaisseau***. Il permet de gérer tous les aspects liés au tir des projectiles. Il s'occupe donc :

- de l'instanciation du projectile ;
- de la propulsion initiale du projectile (application d'une force de type **impulsion**) ;
- de la gestion du temps de recharge ; et,
- de la gestion du nombre de projectiles actifs.

Ce script contiendra plusieurs méthodes, dont deux méthodes publiques :

- La méthode **`Tirer()`** qui sera appelée par le script ***InputManager*** qui gère les commandes effectuées par le joueur; et,
- La méthode **`DétruireProjectile()`** qui sera appelée par le script ***ActionProjectile*** lorsque ce dernier désirera détruire un projectile.

10. Script ComportementVaisseau (à compléter)

Ce script est un composant du ***Prefab Vaisseau***. Il permet de gérer tous les aspects liés aux collisions du vaisseau avec les autres ***GameObject*** de la scène.

a) Dans le cas d'une collision avec une caisse ou une cible, le joueur recevra une pénalité de 5 points qui devra évidemment s'afficher sur le panneau de pointage. Cependant, ni la caisse ou la cible ni le vaisseau ne sont détruit.

b) Dans le cas d'une collision avec le ***prefab*** symbolisant la destination, il faudra alors faire appel à la méthode **`TrouverDestination()`** du script ***FsmJeu***. Ce script comporte également la méthode publique **`InitialiserComportementVaisseau()`** qui permettra d'établir les liens entre ce script et les scripts ***GestionPointage*** et ***FsmJeu***.

11. Script InputManager (déjà rédigé)

Ce script permet de gérer les entrées au clavier effectuées par le joueur et d'activer les fonctionnalités demandées. Vous remarquerez que les touches et les fonctionnalités sont modifiables à partir de l'éditeur d'Unity.

12. Script MouvementVaisseau (à compléter)

Ce script est un composant du ***Prefab Vaisseau***. Il permet de gérer tous les aspects liés aux mouvements du vaisseau. Ce script va ainsi gérer les aspects liés à la simulation physique qui implique l'application de force sur le vaisseau. Il va aussi gérer certains aspects liés à la simulation visuelle en simulant « l'allumage » des réacteurs et des rétrofusées.

Pour ce script, vous devrez compléter certaines méthodes d'initialisation, ainsi que l'ensemble des méthodes liées au mouvement du vaisseau.

13. Bonus : Script ComportementAtome (à compléter)

Ce script est un composant du ***Prefab Atome***. Il s'occupe de déterminer le nombre d'électrons et de les instancier. Pour ce script, vous devrez compléter la méthode d'initialisation.

14. Bonus : Script ComportementÉlectron (à compléter)

Ce script est un composant du ***Prefab Électron***. Il s'occupe de l'initialisation et du mouvement circulaire de l'électron. Pour ce script, vous devrez compléter la méthode d'initialisation et celle de la mise à jour.

15. Bonus : Script ComportementMine (à compléter)

Ce script est un composant du ***Prefab Mine***. Il s'occupe tout d'abord de l'initialisation de la gestion de l'intensité lumineuse et de l'initialisation de l'explosion de la mine. Ensuite, il gère la fluctuation de l'intensité lumineuse et la réaction du déclencheur. Pour ce script, vous devrez compléter la méthode de mise à jour, celle qui gère la réaction du déclencheur ainsi que celle qui permet d'initialiser le lien entre la mine et le script ***FsmJeu*** de l'objet ***GameManager***.

D. Conditions de réalisation et de remise

a) Ce travail pratique **doit** être réalisé en équipe de deux. Le projet doit être divisé équitablement et intelligemment entre les deux membres de l'équipe pour qu'en cas de problème on soit capable de distinguer ce que chacun a effectué comme travail. Vous devez conserver une trace du travail réellement effectué par chacun.

b) Vous devez partir du projet de départ disponible sur le site Moodle du cours en sélectionnant le projet qui correspond à la taille de votre équipe.

c) Vous devez absolument respecter la nomenclature des fichiers et celle des attributs exposés dans l'éditeur, incluant l'usage des accents. Vous serez pénalisé si vous ne le faites pas, car vos scripts provoqueront des erreurs lors de l'exécution du programme de test. Vous ne pouvez pas ajouter de nouveaux scripts ou de nouvelles classes.

d) Vous devez indiquer au sommet de chaque script le nom des étudiantes ou des étudiants qui ont réellement travaillé sur ce script.

e) Vous devrez remettre, dans la boîte de remise associée à l'évaluation sur le site Moodle, un fichier d'archives contenant tous les scripts du projet.

f) Vous devez nommer ce fichier d'archives (ZIP) en respectant les règles de toponymie suivante : « ***TP2 - Nom1 Prénom1 - Nom2 Prénom2*** »

g) Ce fichier d'archives devra être remis **avant minuit** le **mercredi 16 octobre** prochain.