

Cégep de Sainte-Foy – Automne 2025
Programmation de jeux vidéo II – 420-W51-SF

Travail Pratique 1

30 septembre 2025

Préparé par
Benjamin Lemelin et François Paradis

1 Résumé

Réaliser un jeu de tir à la troisième personne dans Unity. Dans ce jeu, des aliens viennent envahir l'univers du joueur. Le joueur doit éliminer les aliens et détruire tous les portails où ils apparaissent pour gagner.

Valeur de la note finale	Contexte	Durée
20%	En équipe de 2	2 ½ semaines

2 Tâches à réaliser

2.1. Explorer le projet de départ

Importez le projet de départ fourni dans Unity et ouvrez la scène nommée **Game**. Cette scène contient déjà plusieurs objets servant de point de départ à votre jeu. Entre autres :

- **Arena** : Monde du jeu. Les *colliders* sont déjà configurés, donc en temps normal vous ne devriez pas avoir à le toucher (hormis lui ajouter une surface navigable pour les aliens).
- **SpaceMarine** : Personnage joueur. Possède le tag **Player** (nécessaire au **Finder**). Contient aussi un script pour ses animations (vous ne devriez pas toucher à ce script).
- **Alien** : Ennemis du jeu. En dehors du visuel, ainsi que des animations, il n'y a rien de préconfiguré sur ceux-ci. Vous aurez à ajouter la logique, les collisions, la navigation, etc.
- **Portal** : Portail d'où apparaissent les aliens. En temps normal, cet objet devrait rester très simple. Vous aurez à ajouter leur logique ainsi que les collisions.
- **Bullet** : Projectile que le joueur peut tirer. Encore une fois, en dehors du visuel, rien n'a été configuré sur cet objet : il vous faudra le programmer au complet et ajouter les collisions.

Vous trouverez aussi dans le dossier **Prefabs** les éléments suivants :

- **Missiles** : Projectiles spéciaux que le joueur peut tirer. Ils font plus de dégâts que les projectiles normaux. Notez qu'il y a un *prefab* nommé **BaseMissile** dont tous les autres héritent.
- **Collectible** : Objets que le joueur peut ramasser, chacun ayant un effet différent (tel que donner des munitions). Notez qu'il y a aussi un *prefab* nommé **BaseCollectible** dont les autres héritent.
- **FX** : Effets de particules variés. Hormis quelques cas spécifiques, vous n'aurez pas besoin de la majorité d'entre eux. Cela dit, vous pouvez en utiliser plus comme bonus.

Ouvrez le projet C# avec Unity. Ce projet contient déjà plusieurs scripts, dont un nommé **ObjectPool** (vous en connaissez l'utilité). Tous les autres scripts sont à créer par vous-même.

Important

Il y a un dossier nommé **Util**, mais il vous est **strictement interdit** d'y créer quoi que ce soit. Classez vos scripts en fonction de l'entité (par exemple, ce qui touche aux aliens va dans un dossier **Alien**).

2.2. Space Marine

Représente le joueur. Il peut se déplacer et aussi sauter (mais pas de double saut). Il peut aussi tirer des [projectiles](#) à volonté (mais pas les [missiles](#)). Lorsqu'il prend des dégâts, il devient invulnérable pendant un court laps de temps. Il meurt si ses points de vie tombent à 0 (voir la section sur la fin de partie).



Métriques	
Vitesse de déplacement	25 u/s
Vitesse de rotation	30 u/s
Hauteur du saut	10 u
Points de vie au début du jeu	50 pts
Nombre maximal de points de vie	100 pts
Durée d'invulnérabilité	1.5 secondes







Voici les paramètres pour un **CapsuleCollider** ou un **CharacterController** :

Paramètres	
Centre	X : 0.0 Y : 5.7 Z : 0.0
Rayon	3
Hauteur	11

⚠ Important

Même si vous avez un **CharacterController**, vous aurez aussi besoin d'un *collider* afin de détecter les collisions entre le joueur et les aliens. N'ajoutez pas de **Rigidbody** : cela crée des problèmes.

Voici le schéma des contrôles à utiliser. Votre jeu doit supporter à la fois le clavier/souris et les manettes.

Action	Clavier/Souris	Manette
Se déplacer	W A S D	
Bouger la caméra		
Sauter	SPACE	
Tirer un projectile		R
Tirer un missile		L

i Information

Dans le *New Input System* de Unity, assignez l'entrée **Mouse/Delta** au mouvement de la caméra pour le combo clavier/souris. Autrement, vous n'arriverez pas à le faire marcher avec *Cinemachine*.

2.3. Aliens

Représentent les ennemis. Ils surgissent des portails et se mettent à poursuivre le joueur (ils doivent éviter les obstacles). Si un alien entre en contact avec le joueur, il lui fait des dégâts et meurt. Les [projectiles](#) du joueur le tuent instantanément (que ce soit le projectile de base ou l'explosion d'un missile).



Métriques	
Vitesse de déplacement	10 u/s
Vitesse de rotation	120 u/s
Dégâts appliqués au joueur	10 pts

Voici les paramètres pour un **CapsuleCollider** ou un **NavMeshAgent** :

Paramètres	
Centre	X : 0.0 Y : 3 Z : 0.0
Rayon	3
Hauteur	7
Hauteur des marches	2

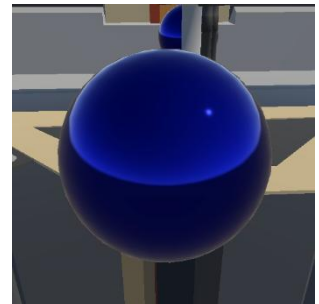
⚠ Important

Ajoutez un **Rigidbody** aux aliens pour les collisions avec le joueur. Dans ses contraintes, verrouillez les rotations **XYZ**. Aussi, utilisez la géométrie des *Physics Collider* pour calculer le **NavMeshSurface**.

2.4. Portails

Point d'apparition des aliens. Ces portails sont disposés un peu partout dans le monde du joueur (placez-en au moins 5). Un alien sort de l'un de ces portails à toutes les 0.5 secondes pour attaquer le joueur. Autrement dit, à toutes les 0.5 secondes, 1 alien sort de 1 portail choisi aléatoirement parmi ceux restants.

Le joueur peut détruire un portail en tirant dessus pour qu'il arrête de générer des aliens. La destruction d'un portail crée aussi un [collectible](#) aléatoirement.



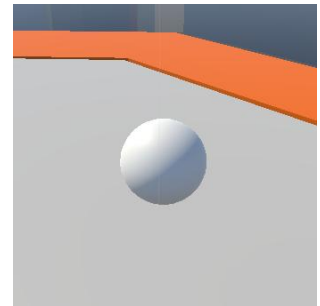
Métriques	
Points de vie	10 pts
Délai de création d'alien	0.5 seconde
Nombre d'alien maximal dans le monde	20

💡 Astuce

Ayez une liste des portails actifs dans le **GameController**. Lorsqu'un portail s'active (**OnEnable**), ajoutez-le dans la liste. Lorsqu'il se désactive (**OnDisable**), retirez-le de la liste.

2.5. Projectiles

Projectiles tirés par le joueur. Ces projectiles font des dégâts aux aliens et aux portails. Ils sont aussi détruits s'ils frappent un mur. Les projectiles doivent démarrer leur course à partir du canon de l'arme du joueur (voir le **SpawnPoint**). Le joueur peut tirer des projectiles à l'infini (il n'a pas de limite de munitions).



Métriques	
Vitesse	200 u/s
Dégâts appliqués	1 pt
Inventaire de munitions	Illimité
Rayon du <i>collider</i>	0.5

i Information

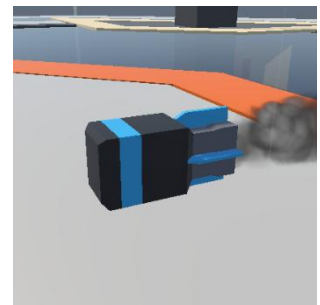
Un projectile ne devrait pas se soucier du type d'objet frappé (alien ou portail). Faites usage de polymorphisme (donc, d'interfaces) pour passer les messages.

! Important

Utilisez un *trigger* pour les projectiles. Autrement, ils pourraient rebondir de manière erratique. N'oubliez pas d'inclure un **RigidBody**, mais vous devriez en désactiver la gravité.

2.6. Missiles

Projectiles plus gros tirés par le joueur. Ils ont les mêmes propriétés que les projectiles normaux, mais font plutôt des dégâts dans un cercle lors de l'impact. Le joueur n'a aucun missile dans son inventaire au début du jeu : il doit les récolter dans les *collectibles*. La couleur du missile est purement esthétique.



Métriques	
Vitesse	200 u/s
Dégâts appliqués	10 pt
Taille de la zone de dégâts	20
Inventaire au début du jeu	0
Inventaire maximal	10
Rayon du <i>collider</i>	0.7

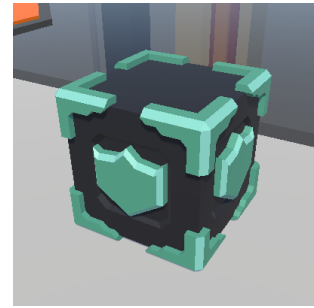
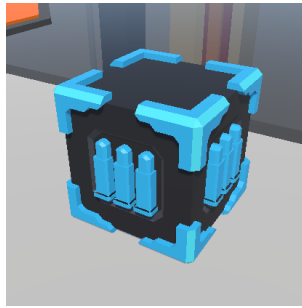
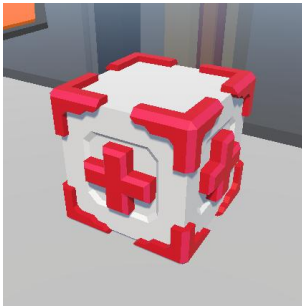
i Information

Lorsqu'une collision est détectée, utilisez la fonction **Physics.OverlapSphere** pour [détecter les objets](#) autour du projectile. Il existe une version [n'allouant pas de mémoire](#), qui est aussi plus performante, mais vous n'avez pas à vous en servir si le temps vous manque.

2.7. Caméra

La caméra doit utiliser *Cinemachine*. Utilisez une caméra de type orbitale tournant autour du joueur. En ce qui concerne le composant **CinemachineInputAxisController**, multipliez le paramètre **Gain** par 10 afin que la caméra soit plus rapide. Tâchez aussi d'assigner le bon *Input Action* dans les paramètres.

2.8. Collectibles



Objets sous forme de boîte pouvant être ramassés par le joueur. Un *collectible* aléatoire doit être créé à l'emplacement de chaque portail détruit. Ils ont différents effets selon le type :

- **Soin** : Restaurent 25 pts de vie au joueur.
- **Munitions** : Donnent 5 missiles au joueur. Le joueur ne peut pas avoir plus de 10 missiles.
- **Armure** : Rendent le joueur invulnérable pendant 5 secondes.

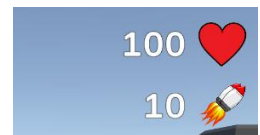
Les *collectibles* disparaissent automatiquement au bout de 10 secondes s'ils ne sont pas ramassés. Tout comme les aliens, vous devez recycler ces objets. Enfin, utilisez un *trigger* pour la détection des collisions.

💡 Astuce

Les *collectibles* risquent de créer une bonne quantité de duplication de code. Dans le domaine du jeu vidéo, ce n'est pas dramatique, mais si vous tenez à l'éliminer, consultez votre professeur.

2.9. Affichage tête-haute (HUD)

Le nombre de points de vie et le nombre de missiles doit être visible dans le coin supérieur droit de l'écran. Les images sont toutes fournies avec le projet dans le dossier **Visuals**. La taille du texte est de 100 (en 1920x1080).



📘 Information

Pour ajouter une bordure au texte, modifiez son **Material Preset** pour la variante **Outline**.

💡 Astuce

Paramétrez le **Canvas Scaler** pour adapter automatiquement le UI à la taille de l'écran

2.10. Victoire

Si tous les portails sont détruits, alors le joueur gagne la partie. Une musique de fin de partie doit jouer et un message doit être affiché à l'écran. La taille du texte est **200** et sa couleur (en hexadécimal) est **#49FF7A**.



Lors de la victoire du joueur, tous les aliens restants doivent mourir. Vous devez **obligatoirement** faire usage d'un canal événementiel (comprendre le patron *Publish/Subscribe*) signaler la fin de la partie.

2.11. Défaite

Si les points de vie du joueur atteignent 0, alors le joueur perd la partie. Une musique de fin de partie doit jouer et un message doit être affiché à l'écran. En guise de bonus, vous pouvez détacher la tâche du *Space Marine* et la faire rouler par terre, mais c'est facultatif. Dans cet écran, la couleur du texte est **#FF4A56**.



2.12. Musique

Une musique doit jouer **en boucle** pendant le jeu. Elle est déjà fournie, mais si vous le souhaitez, vous pouvez aussi en utiliser une autre (si cela reste dans le thème du jeu). Le volume par défaut est de **0.3**.

2.13. Sons

Certains événements produisent un son :

- Mort d'un alien (voir le son **AlienExplosion**).
- Destruction d'un portail (voir le son **PortalExplosion**).
- Explosion d'un missile (voir son **MissileExplosion**).
- Joueur prends des dégâts (voir son **SpaceMarineHurt**).
- Mort du joueur (voir son **SpaceMarineDeath**).
- Tir d'un projectile (voir son **ProjectileShot**).
- Ramasser un *collectible* (voir son **CollectibleCollect**).

Astuce

La majorité des sons jouent lorsqu'un objet est créé (par exemple, le son du tir d'un projectile). Jouez ces sons en même temps que l'objet est activé (voir callback **OnEnable**). Vous pouvez même faire un composant réutilisable nommé **PlaySoundOnEnable** : simple et efficace.

Important

Le son des *collectibles* risque de vous donner du fil à retordre. Lorsqu'ils sont collectés, leur **GameObject** est recyclé, et donc désactivé. Le son ne peut alors plus jouer.

Une stratégie dans ce cas de figure consiste à créer un **AudioSource** global. Vous avez déjà fait ce genre de chose dans la *Machine Pachinko*.

2.14. Effets de particules

Certains événements entraînent la création d'un effet de particule :

- Mort d'un alien (voir *prefab* **AlienExplosion**).
- Destruction d'un portail (voir *prefab* **PortalExplosion**).
- Explosion d'un missile (voir *prefab* **MissileExplosion**).

Comme les aliens et les *collectibles*, vous devez faire usage de recyclage pour les effets de particules. À vous de prévoir un script qui, après un certain temps, recycle l'effet de particule. Vous devez **obligatoirement** le faire via une coroutine. Vous pouvez assumer que chaque effet de particule dure environ 3 secondes (mais il faut que ce soit configurable à partir de l'éditeur si nécessaire).

Astuce

L'effet de particule devra déterminer à quel **ObjectPool** il appartient. Le plus simple est d'utiliser la fonction **GetComponentInParent**, car les instances du *pool* sont des enfants directs.

3 Contraintes

- Le code doit être en **anglais**. Les commentaires peuvent être en français ou en anglais.
- Tout texte dans le jeu (tel que le HUD) doit être en **français**.
- L'utilisation du *New Input System* de Unity est obligatoire.
- Vous devez avoir un **Finder** (ce sera très difficile sans lui de toute façon).
- Vous devez utiliser un **CharacterController** pour déplacer le joueur.
- Vous devez utiliser un **NavMeshAgent** pour déplacer les ennemis.
- Il est strictement interdit d'ajouter des scripts dans le dossier **Util**.

4 Fraude et plagiat

Toute forme de fraude, plagiat ou tricherie sera sévèrement sanctionnée. Cela inclut notamment le fait de s'approprier le travail d'autrui, y compris celui généré par une intelligence artificielle. Le simple fait d'obtenir et même de partager des réponses est automatiquement considéré comme du plagiat.

Tout comportement de ce type entraînera automatiquement l'attribution de la note de 0%. La fraude sera aussi déclarée à la direction des études, qui pourra imposer d'autres sanctions au besoin, tel que l'expulsion permanente du programme ou du collègue.

5 Modalités de remise

Remettez votre projet sur LÉA, dans la section travaux, à l'intérieur d'une archive *Zip*. Une pénalité de 15 % est appliqué en cas de retard de moins d'une journée. Au-delà de ce délai, le travail est refusé et la note de 0 % est automatiquement attribuée.

Supprimez les fichiers et dossiers inutiles, c'est-à-dire :

- Les dossiers **.vs**, **.idea**, **.git**, **Library**, **Logs**, **obj**, **Temp** et **UserSettings**.
- Les fichiers **.sln**, **.csproj**, et **.user**.
- Cet énoncé.

6 Évaluation

Critères	Pondération
Utilisation correcte du C# et des scripts Code clair et algorithmes compréhensibles (documentées si nécessaire). Respect de l'encapsulation. Respects des standards du langage. Code de qualité. Exploitation judicieuse des possibilités du langage (ne pas recoder ce qui existe déjà). Faire des petits scripts reliés à un comportement donné. Bonnes solutions pour la communication entre scripts. (Peut être ajusté si pas suffisamment de contenu).	20 %
Utilisation correcte de Unity et de ses concepts. Utilisation correcte et judicieuse des <i>Game Objects</i> et des composants. Respect de la boucle de jeu Unity et de ses parties. Utilisation judicieuse des <i>colliders</i> , <i>triggers</i> et de la physique. Communication entre <i>Game Objects</i> . Recyclage d'objets et stabilisation de la mémoire. Gestion des entrants (inputs). Utilisation judicieuse du moteur Unity (<i>Project</i> , <i>Hierarchy</i> et <i>Inspector</i>). (Peut être ajusté si pas suffisamment de contenu).	30 %
Partie fonctionnelle. Partie complète, jouable et sans bug. Le <i>fun</i> du gameplay n'est pas évalué, mais une expérience particulièrement désagréable pourrait être pénalisée.	50 %
Points bonus Peuvent varier selon le cas. À négocier d'avance avec le professeur.	+ 15 %

Pénalités – Rigueur
Qualité générale du code (incluant, mais sans s'y limiter) : <ul style="list-style-type: none"> • Propreté générale du code. • Mauvaises pratiques de programmation. • Formatage ou indentation déficiente. • Standard de nommage non respecté. • Cohérence de l'ensemble de l'œuvre. • Orthographe et grammaire (0.5% jusqu'à concurrence de 20 %).
Remise du travail (incluant, mais sans s'y limiter) : <ul style="list-style-type: none"> • Erreur ou avertissement à l'interprétation du code. • Non-respect des consignes de remise. • Fichier inutile ou temporaire remis avec le projet. • Travail remis en retard.

Important

Le professeur se donne le droit de questionner tout étudiant sur le code qu'il a rédigé. La note finale pourra alors être pondérée si les réponses sont jugées insatisfaisantes.