420-W51-SF
PROGRAMMATION DE JEUX VIDÉO II
TRAVAIL PRATIQUE #1 - SHOOTER GAME
PONDÉRATION : 20%

OBJECTIFS PÉDAGOGIQUES

Ce travail vise à familiariser l'étudiante ou l'étudiant avec les objectifs suivants :

- Effectuer le développement d'applications de jeu ou de simulation (compétences 00SW)

De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de programmation, de contrôle qualité et de documentation.

MISE EN CONTEXTE ET MANDAT

Ce travail pratique a pour but premier de créer un jeu complet en 3D (Affichage rendu 3D), sur le principe du "3rd person shooter".

Quelques contraintes, ainsi que certaines caractéristiques des acteurs seront présentées ici. Pour le reste, ce sera aux étudiants mêmes d'implémenter leur version du jeu suivant leurs observations du prototype présenté.

Par la suite, l'étudiant(e) pourra étudier et réutiliser des éléments d'exercices précédents pour accélérer le développement du jeu.

Finalement, dans un environnement C# / Unity, l'étudiant devra développer petit jeu complet et en assurer la qualité à la fois par des tests d'utilisation.

CONTRAINTES DE DÉVELOPPEMENT

- Classez vos assets de manière approprié dans la fenêtre de projet.
- Gardez votre fenêtre de hiérarchie propre et bien classée.
- Si vous avez plusieurs Game Objects identiques vous devez implémenter des Game Objects de regroupement (pensez à mettre la position de ces objets de regroupement à 0,0,0)
- Il n'y a jamais de bonne raison de ne pas faire de prefabs. Faites de vos Game Objects des prefabs autant que possible.
- Favorisez les petits scripts courts. Pour un acteur donné, vous deviez implémenter un script par comportement. Par exemple votre space marine devrait en avoir au pour le déplacement, un pour le tir, un pour la gestion de la vie et des dommages (et certains de ceux-ci pourraient être encore divisés). N'oubliez pas qu'il est facile de communiquer entre components de script d'un même Game Object (getComponent).
- La stabilisation de la mémoire et le recyclage d'objet doivent être implémentés. Tous les objets de la scène de jeu doivent être présents en mémoire en début de partie. Faites vos instanciations dans le Start de vos scripts (ou dans le awake si ce sont des initialisations purement locales au GameObject courant). Aucun instantiate n'est permis à partir de l'update ou du fixed update.
- Utilisez correctement les méthodes de la boucle de jeu de Unity (qu'est-ce qui va dans l'update et qu'est ce qui va dans le fixed update? Etc.)
- Le jeu doit pouvoir se jouer se jouer au gamepad et au clavier-souris. Le Input System DOIT être utilisé.

SPÉCIFICATIONS DE L'APPLICATION DE BASE

Vous allez recréer une version du petit jeu « ShooterGame » présenté en exécutable. Vous pouvez utiliser sans limite tout le code que vous pourriez trouver dans les différents corrigés et démonstrations associés au cours. Tout autre bloc de code trouvé sur le web peut également être utilisé, mais sa provenance doit être indiquée en commentaire.

- En cas de questionnement sur telle ou telle fonctionnalité, vous devez bien entendu consulter le professeur, mais de manière plus immédiate, vous pouvez vous fier au déroulement de la démo fournie. **Les metrics du jeu présenté sont des exemples éminemment modifiables. Vous pouvez les modifier tant que le principe du jeu reste bon!**

- Vous aurez un projet de départ. Dans ce projet, vous avez l'arène de départ, ainsi qu'un modèle pour le space marine du joueur et un alien. Quelques autres prefabs sont disponibles, et pour la majorité, ils ont peu ou pas de composants attachés. Ce sera à vous de choisir et d'attacher les composants appropriés au rôle et à la situation de chaque Game Object.

- À la base, vous devez implémenter la partie telle que présentée. L'aréna de départ doit contenir quelques obstacles. Les piliers fournis dans le projet de départ seront suffisants, bien que vous puissiez modifier le layout de la map, tant que le tout reste jouable. Quelques portails sont suspendus dans le ciel. Le joueur doit pouvoir entrer en collision avec ces portails et devra sauter pour pouvoir leur tirer dessus et les détruire.

- Des aliens surgissent constamment de ces portails, tombent au sol et se mettent à poursuivre le joueur.

- Un alien meurt au contact du joueur (le space marine se défend agressivement au corps à corps) mais fait un point de dommage au space marine en même temps.

- En poursuivant le joueur, les aliens savent éviter les obstacles et s'éviter entre eux (NavMesh).

- Le space marine peut tirer sur les aliens. Sa cadence de tir est relativement élevée.

- Le space marine peut sauter. Pas de double saut.

- Il peut sauter sur la tête des aliens. En sautant sur un alien, il peut l'abattre sans prendre de dommages (notre space marine est un grand admirateur de Mario! – It's a me! Da Space Mariné! Yahoo!)

- Les aliens peuvent faire tomber des collectibles. Une boite de soin qui redonne un point de vie, une mitrailleuse qui augmente la cadence de tir pendant 10 secondes et des missiles qui font du dommage dans un certain rayon. Les collectibles disparaissent au bout de 15 secondes s'ils ne sont pas récoltés.

- Quand tous les aliens sont vaincus et que tous les portails sont détruits, la musique en court fade out, puis une petite musique de victoire joue, accompagnée d'un message de victoire.

- Appuyer sur Escape ou sur le bouton "menus/back" du gamepad met fin au jeu (dans le jeu et dans l'éditeur : à vous de trouver comment).

Notez que dans les pages suivantes : les points en gras sont des points qui ont été moins bien réussi par le passé. Faites bien attention à ces détails.

Le Space marine

- Sa vitesse de déplacement doit être exposée dans l'inspecteur.
- Son nombre de points de vies doit être exposé dans l'inspecteur.
- Il n'y a pas d'assets de projectiles pour le moment, mais de créer un sphère et d'ajuster la couleur de son material devrait être à votre portée.
- Le tir de projectile doit se faire à partir du canon de l'arme (La matrice de collision pourrait aider ici). Notez que pour la direction du tir, vous pouvez vous aider en obtenant le transform du parent le plus élevé de la hiérarchie courante avec **transform.root**. Mais clairement, pour ce qui est du tir, un peu de recherche sur comment faire les choses pourrait s'avérer nécessaire. **Sachez que le tir devrait suivre la direction de transform.root.forward.**
- **Les projectiles doivent utiliser la vélocité. C'est facile à implémenter et une fois qu'un projectile a une vélocité on peut le laisser aller. L'utilisation de translates ou de moveForward a été source d'un nombre incalculable de bugs par le passé.**
- **Attention : le space marine ne peut pas tirer des projectiles de base et des missiles en même temps. Si la touche de tir de projectile est déjà enfoncée, celle de missile ne provoque aucun tir, et vice-versa.**
- Quand il prend du dommage, il dispose d'une courte période où il est invincible (une demi-seconde dans l'exemple).
- Il élimine les aliens au contact (il est très coriace au corps à corps), et ne prend pas de dommage s'il les élimine en leur sautant dessus (doit arriver assez franc sur le dessus de l'alien).
- Le Space Marine doit avoir un Character Controller et pas de RigidBody. Vous pouvez ajouter un trigger au space marine (qui aura exactement la même forme que le collider du Character Controller) si ça peut vous aider à faire certaines détections.
- Pour le reste, fiez-vous à l'exemple.

L'alien

- Ses points de vies doivent être exposé dans l'inspecteur. Dans l'exemple présent, il meurt d'un coup (un seul point de vie), mais ça pourrait être plus que ça. Néanmoins, il perd toujours tous ses points de vie au contact du joueur.
- Doit pouvoir éviter les obstacles, ainsi que ses copains aliens. Le navmesh est de mise.
- **Notez que dès que le NavMesh Agent est actif, l'alien descend immédiatement au sol (carrément téléporté) et colle dessus. Le NavMesh Agent devra donc être inactif au départ et être activé au moment approprié.**
- L'alien doit spawner dans un spawner au hasard. Puis descend au sol.
- Les aliens ne doivent jamais avoir de "clipping de rendering" entre eux.
- Un alien de doit pas chambouler n'importe comment à la moindre collision et doit rester toujours relativement droit. **Pensez à geler certaines rotations au niveau du rigidBody.**
- Pour le reste, fiez-vous à l'exemple.

Le spawing d'aliens et les Spawn points

- Les spawn points ont des points de vies qui doivent être exposés dans l'inspecteur. Ils devraient avoir plusieurs points de vie (10 dans l'exemple exécutable)
- Il doit y en avoir plus ou moins dix. Ils n'ont pas d'assets pour le moment, mais comme pour les projectiles du Space Marine : sphère et changement de couleur du material.
- Les aliens doivent spawner dans les spawn points et descendre vers le sol (pas de collision entre les spawn points et les aliens - matrice de collision).
- **Par contre, rappel : le space marine, lui doit absolument pouvoir entrer en collision avec les spawners.**
- Les spawn points sont assez surélevés pour que l'alien puisse descendre et que le space marine doive sauter pour pouvoir tirer dessus.
- **(Pour les points suivants : les spawns points sont les sources de spawning. Toute la logique de spawning devrait plutôt être gérée au niveau d'un manager, surtout sachant que des spawn points peuvent disparaitre et que le spawning doit pouvoir continuer avec fluidité peu importe le nombre de spawn points restants).**
- Le temps entre chaque spawn d'Alien est fixe, et doit être exposé dans l'inspecteur, tout comme le nombre maximum d'aliens en jeu en même temps, qui doit être aussi exposé dans l'inspecteur.
- Un nombre maximal d'aliens spawnés doit être exposé dans l'inspecteur. Une fois que le nombre maximum de spawn sera atteint, les spawn points ne produiront plus d'aliens.
- Par exemple, dans la démo présentée, les aliens en jeu sont de 20 et le maximum d'aliens qui peuvent être spawnés sont de 500.
- Un alien peut spawner au hasard dans n'importe quel spawn point toujours actif.

La caméra

- Doit être implémenté selon le mode « FreeLook Camera » présentée dans le tutoriel Cinemachine (en mode totalement libre (World Space) ou avec un léger suivit (Lazy Follow) à votre choix*). Vous pouvez suivre la même méthode vue en classe, bien que la taille des orbites sera à revoir. ALTRENATIVEMENT : Vous pouvez mettre en place une caméra qui reste toujours de dos au joueur, mais ça va vous demander de fouiller le sujet par vous-même.
- La caméra être relativement efficace. Cependant une certaine marge d'erreur sera tolérée ici. La caméra est un élément complexe à mettre en place et peut demander beaucoup de raffinage et d'essai et d'erreur. Ne mettez pas trop d'énergie sur ce point.

* Dans la démo, c'est World Space qui fut utilisé. Lazy follow est meilleure strictement avec le gamepad dans la séquence contre les aliens. Avec clavier-souris sous toute condition et au gamepad contre le robot, la caméra libre est la plus appropriée.

L'arena et le navmesh

- Vous pouvez ajouter des obstacles ou modifier ceux qui sont déjà présents. L'Arena n'a pas du tout à être identique à ce qui est présenté en exemple et peut être bien moins symétrique. Néanmoins les situations où le space marine serait totalement à l'abri des aliens sont à éviter.

- Attention à ce que vous allez sélectionner pour votre navmesh. Regardez les options du Navmesh Surface (Object Collection->Include Layers) et sachez pas que vous pouvez sélectionner seulement certains layers physiques pour générer le navmesh. Les planchers ont déjà leur propre layer; il suffirait seulement que les colonnes aient le leur aussi…

LE HUD

- Doit au moins comprendre les informations présentées dans l'exemple.
- Les icones sont à favoriser plutôt que le texte pour chaque information à transmettre au joueur (comme dans l'exemple).

LES COLLECTIBLES

- Un collectible a une chance sur X d'apparaitre quand un alien est détruit. X doit être exposé dans l'inspecteur (X vaut 15 dans l'exemple fourni). De là, n'importe quel des trois types de collectible doit pouvoir apparaitre au hasard.
- Les collectibles doivent pivoter sur place. Ils ont une durée de vie de 15 secondes.
- Seul le joueur doit pouvoir les récolter. Les autres acteurs peuvent passer au travers.
- Le heal redonne un point de vie au space marine et ça peut aller au-dessus du total de départ.
- Le multishot permet un triple tir pendant 10 secondes. Ramasser un autre triple tir alors que celui-ci est actif ajoute 10 secondes à l'activation.
- Si le tir de base suit la direction du transform.forward du gameObject parent alors les deux diagonales se calculent facilement à l'aide de "transform.forward * 0.5f + transform .right * 0.5f" pour le côté droit et de "transform.forward * 0.5f - transform .right * 0.5f" pour le côté gauche.
- La caisse de missiles donne doit à 5 missiles. Un missile fait perdre 5 points de vie à sa cible.
- Un missile qui entre en collision avec quoi que ce soit explose et fait du dommage à tout acteur ennemi qui a des points de vie aux alentours (alien, spawner, robot si implémenté). Faites apparaitre une sphère rouge pendant une faction de seconde pour simuler une explosion. Il est alternativement possible d'utiliser un effet de particule d'explosion (voir section sur les bonus), mais cet effet ne provoque aucune collision. À vous de trouver une solution si vous utilisez les particules…
- **Pour implémenter la chose, vous n'avez pas de grand calcul de trigonométrie et de calcul de distance à faire; la solution se trouve au niveau des colliders et de leurs attributs. Sachez qu'un component peut être activé et désactivé et que la taille des colliders est aussi un attribut qui peut changer…**

LES SONS

- Tous les sons dans les assets présents doivent être utilisés, sauf la musique : il y en a deux, ainsi que la mort du space marine qui a aussi deux sons – dans les deux cas, choisissez-en un. Le son de l'élévateur n'est bien entendu pas obligatoire non plus.
- Le moment d'utiliser tel ou tel son est assez auto-explicative dans leurs noms. Encore une fois, fiez-vous à la démo exécutable.

- Chaque acteur devrait avoir un seul audio source.
- Un acteur qui est désactivé ne peut plus jouer de son. Toutefois le son de sa mort, ou tout autre son pertinent devra être tout de même être joué à partir de sa position. **À vous de trouver une astuce pour implémenter cela (une action reliée à un GameObject ne doit pas nécessairement être faite par lui).**
- Quand les aliens sont vaincus la musique fait un fade-out de 3 secondes, puis la musique de victoire joue, sans mise en boucle.

CES ÉLÉMENTS PRÉSENTS DANS LA DÉMO NE SONT PAS À FAIRE… MAIS POURRAIT ÊTRE FAIT ET RAPPORTER DES POINTS BONIS.

- Le cheat mode.
- L'animation du space marine et de l'alien (pour l'alien l'animation de base est très facile à faire et rapportera peu, par contre si l'alien cours seulement quand il est au sol et que sa vitesse d'animation est relative à sa vitesse de déplacement, alors là ça va payer).
- La mort du space marine où la tête se sépare du corps (normalement, la simple désactivation du space marine pour représenter sa mort est suffisante).
- Des effets de particules pour la mort des aliens, la trainée des missiles et les explosions de missiles. Notez que tous ces assets sont déjà présents dans le projet et même déjà présents sur les acteurs en question. À vous de trouver comment les utiliser.
- La dynamique de combat du robot (majeur).
- Tout autre élément supplémentaire que vous souhaiteriez implémenter (discutez-en **obligatoirement** avec votre professeur avant de vous lancer).
- Chaque élément va rapporter des points bonus selon la quantité et complexité du travail fourni. Le bonus maximal est de 15% et la note maximale reste de 100%.

LE ROBOT

- **N'est pas à implémenter (mais serait un bonus si vous le faites quand même).**
- A un nombre de points de vies exposé dans l'inspecteur (100 dans l'exemple).
- Descend de son point surélevé puis entre en action quand tous les aliens et les portails ont été éliminés.
- Circule au hasard entre différents points d'intérêts, tout en faisant face au joueur.
- Tir vers le joueur projectile et missiles. Les missiles font 5 points de dégâts. Le contact avec le robot fait 5 de dégâts.
- Une fois le robot vaincu, le space marine gagne la partie.
- Le dernier point de la sections "sons" est remplacée par ceci :
  "Quand les aliens sont vaincus la première musique fait un fade-out de 3 secondes, puis la seconde musique embarque pour le combat contre le robot. Suite au combat contre le robot, quand celui-ci est vaincu, on a un nouveau fade-out et la musique de victoire joue, sans mise en boucle."

ASSETS GRAPHIQUES ET SONORES

Des assets graphiques vous sont proposés. Cela dit, rien ne vous interdit d'utiliser des alternatives si vous le souhaitez. L'esthétique graphique ne sera pas évaluée (bien qu'un rendu agressant ou de piètre qualité puisse être pénalisée).

À l'exception des spawn points et des projectiles, qui sont de simples sphères avec un matériel brillant et reflétant, tous les assets sont déjà présents dans le projet.

Il est possible de remplacer certains sons si vous le souhaitez. Mais tous les sons présents (ou vos alternatives) doivent être utilisés. Dans tous les cas, consultez votre professeur si vous voulez être certain que votre alternative est acceptable.

CONSEILS DE DÉVELOPPEMENT.

Faire le mapping correct de vos contrôles avec l'Input System. Utiliser les gâchettes de la manette est très simple. **Utilisez les mêmes mappings que l'exemple.**

Le character controller est de mise pour le space marine, mais sa gestion de collision est plutôt particulière (voir notes du cours 05). Vous pouvez parfaitement attacher un trigger au space marine qui aurait la même taille que le collider du character controller si ça peut vous aider. Aussi **mettez Min Move Distance à 0** (pour améliorer la détection de collision) et **Step Offset à 1**; le plancher n'est pas parfaitement plat, et par défaut, le marine verra certaines différences de hauteur de plancher comme des obstacles infranchissables.

Encore une fois, le recyclage d'objet est de mise. Faites vos pools d'objets et désactivez-les en début de scène. OnEnable sera d'un très grand aide. Écrivez toute de suite cette méthode dans les scripts des objets qui seront amenées à être réinitialisés lors du recyclage, ça va vous rappeler que vous devez l'utiliser.

Il y a aura de nombreuses collisions/triggers dans ce jeu. Vous pouvez utiliser les tags pour distinguer quel type d'acteur touche quel autre type d'acteur, mais attention chaque acteur a droit à un seul tag.

La matrice de collision est votre amie. Certains acteurs ne devraient jamais collisionner avec d'autres types d'acteurs, à commencer par un acteur qui tire des projectiles : pas de collision possible entre l'acteur et ses propres projectiles. Le joueur a aussi deux types de projectile. On ne devrait pas pouvoir détruire nos propres missiles avec nos bullets.

Utilisez la vélocité pour vos projectiles. Une fois une vélocité appliquée, on a plus besoin de s'occuper de rien. Translate / moveTo / etc. sont vivement déconseillés, étant sources de plus de bugs que vous pourriez l'imaginer, et demandant d'appliquer le mouvement à chaque update.

CONTEXTE DE RÉALISATION ET DÉMARCHE DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **en équipe de deux. Consultez votre professeur si une difficulté majeure vous empêcherait de faire le travail en équipe.**

Biens livrables :

- Projet nettoyé de votre application (Assets/Packages/Project Settings) si vous déposez votre projet sur LÉA. Alternativement, vous pouvez publier le lien vers un dépôt Git, Google ou autre, où se trouve votre projet. Si tel est votre choix, assurez-vous que le projet est téléchargeable par un tier.

- Un log de tâche décrivant quel membre a travaillé sur quelle tâche. Le but étant surtout de s'assurer que le travail ai bien été équitable entre les deux membres. Ce log sera renvoyé avec le corrigé du TP.

Conditions de remise :

- L'application doit compiler et être fonctionnelle (On ne se battra pas longtemps pour faire fonctionner votre projet s'il ne s'exécute pas dans Unity au départ.).

Date de remise : **21 octobre en fin de journée.**

TRAVAIL D'ÉQUIPE

Si les équipiers travaillent dans des proportions similaires, les deux étudiants devraient obtenir la même note.

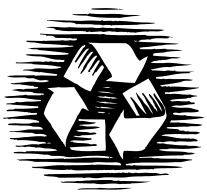
Il est du devoir de chaque membre d'équipe de contre-vérifier le travail du partenaire et de s'assurer de la qualité de ce qui est livré. Le mandat est un mandat commun.

En cas de conflit, le professeur peut convoquer une équipe afin de départager le travail fait et ajuster les points en conséquence (le log de tâches devrait aider en ce point précis).

MODALITÉS D'ÉVALUATION

- Tous les **biens livrables** devront être **remis à temps** et selon les modalités spécifiées.

- Dans l'éventualité où vous récupériez du code existant ailleurs (internet, MSDN, etc), vous devez clairement indiquer la source ainsi que la section de code en question. Tout travail plagié ou tout code récupéré d'une source externe et non mentionnée peut entraîner la note zéro (0) pour l'ensemble du travail.



Grille d'évaluation du travail pratique #1	
Critères d'évaluation	
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 mitigé si pas suffisamment de contenu).	**20%**
Utilisation correcte de Unity et de ses concepts. Utilisation correct et judicieuse des Game Objects et des composants. Respect de la boucle de jeu Unity et de ses parties. Utilisation judicieuse des colliders, triggers et de la physique (y compris la matrice de collision). Communication entre Game Objets. Recyclage d'objets et stabilisation de la mémoire. Gestion des entrants (inputs). Utilisation judicieuse de l'interface de Unity (Project, Hierarchy et Inspector). (Peut être mitigé si pas suffisamment de contenu).	**30%**
Partie fonctionnelle. Partie complète, jouable et sans bug. Le fun du gameplay n'est pas évalué, mais une expérience particulièrement désagréable sur un aspect donné pourrait être pénalisée.	**50%**
Bonus (voir section plus haut)	**+15%**
TOTAL	**100%**