# 420-V31-SF
# PROGRAMMATION DE JEUX VIDÉO III
# TRAVAIL PRATIQUE #1
# ENDLESS ZOMBIE
# PONDÉRATION : 10%

## OBJECTIFS PÉDAGOGIQUES

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

- Appliquer une approche de développement par objets (compétences 016T)
- Apporter des améliorations fonctionnelles à une application (compétence 0176)
- Concevoir et développer une application dans un environnement graphique (compétence 017C)

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

## MISE EN CONTEXTE ET MANDAT

Ce travail pratique a pour but premier de créer un jeu complet en 2D (Affichage par Sprite), sur le principe du 'Twin Stick Shooter'.

L'étudiant devra, suite aux spécifications de l'application, définir une architecture orientée objet à l'aide de langage UML. Il pourra par la suite utiliser des éléments de code d'exercices précédents pour accélérer le développement du jeu.

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

## SPÉCIFICATIONS DE L'APPLICATION DE BASE

***Vous allez recréer une version du jeu 'I made a Game with Zombies in It'*** (*https://www.youtube.com/watch?v=v5T6qenbdUc*) ***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 du jeu original. **Les metrics sont founis en exemple. Vous pouvez les modifier tant que le principe reste bon!**
- À la base, vous devez implémenter la partie telle que présentée. Une carte de jeu avec un tireur et plusieurs zombies apparaissant sur la carte et convergeant toujours vers un personnage qui tente de survivre. Tel que mentionné, le principe du 'Twin Stick Shooter' s'applique ici. Pour un gamepad, le joystick de gauche pour le mouvement, le joystick de droit pour l'orientation. Sans gamepad, les touches du clavier guident la direction du tireur de zombies et la position de la souris guide son orientation.

- Vous aurez un projet de départ. Dans ce projet, quelques sprites seront affichés comme référence et il sera possible de se déplacer à l'aide des touches du clavier dans un espace plus grand que l'écran. Cet espace devrait être l'espace de jeu défini.
- À la base, le tireur possède un nombre illimité de munitions. Les zombies sont tués dès qu'ils sont atteints par un projectile. Le tireur meurt s'il est atteint par un zombie.
- Le joueur a cinq vies et en gagne une nouvelle par tranche de 50000 points. Si le joueur est tué, les zombies s'éloignent de l'endroit où le joueur est mort pendant deux secondes, puis, le joueur re-spawn à l'endroit même où il a été touché et les zombies se remettent à avancer vers le joueur. Le joueur est invincible pendant 3 secondes (mettre un peu de transparence sur le joueur pour illustrer cela).
- Vous pouvez acquérir des armes spéciales. Des icônes de munitions arrivent au hasard quand un zombie est abattu. Si le joueur saisit des munitions, celles-ci s'ajoutent et l'arme spéciale est automatiquement utilisée. Si la même munition est saisie, celle-ci s'ajoute aux munitions restantes. Si les munitions d'une nouvelle arme est saisie, celles déjà présentes sont réduites à zéro et les nouvelles munitions sont ajoutées. Si les munitions spéciales tombent à zéro, on revient au tir de base.
- Voici les armes (les rythmes de tir sont indicatifs, vous pouvez les changer à loisir)
    - Tir de base : Petit projectile rapide. Élimine un zombie au contact et le projectile est lui aussi éliminé. Tir en continue possible. 6 à 8 tirs par seconde.
    - Spray shoot : 3 tirs de base en éventail. 3 ou 4 tirs par seconde (+60 par icône)
    - Lance-Flamme : Projectile rond, massif, très lent et qui ne "meurt" pas au contact des zombies, mais après environ une seconde. Ne voyage pas dans tout l'écran (+250 par icône)
    - Roquette : traverse tout l'écran et élimine tous les zombies qu'elle rencontre. Deux tirs à la seconde (+50 par icône).
- 100 points par zombie tué. Le zombie suivant vaut +10 s'il est tué dans la demi-seconde suivante, et ainsi de suite, jusqu'à un maximum de 500 points. Dès qu'une demi-seconde est écoulé depuis le dernier "kill" le zombie suivant vaut 100 points à nouveau.
- Nombre de zombies maximum suggérés : entre 25 et 50. Utilisez un tableau fixe, et placez-y directement vos zombies. Déterminez un état actif / inactif, mais pour simplification, le zombie devrait toujours être en mémoire.
- Vos projectiles devraient aussi être "recyclés"
- Un zombie ou un bonus devraient spawner à une distance minimale du joueur, (50 à 100 pixels seraient un bon minimum). Un bon modèle est l'algorithme de téléportation du Pokémon dans le solutionnaire de Pokémon 2. Des zombies peuvent spawner hors de la vue.

## CONTRAINTES DE DÉVELOPPEMENT

- Dans les méthodes, toujours passer les objets par référence.

- Utilisation systématique de const, à la fin des méthodes qui ne modifie pas l'objet courant (dont les get) et const sur les paramètres en entrée que l'on ne veut pas modifier (donc sur la grande majorité des set).

- Un maximum de code sera dans les classes appropriées. Si vous avez le choix entre mettre du code dans une classe spécifique et du code dans la classe game, le premier choix est le bon, à l'exception notable du chargement des ressources. Vous pouvez garder les ressources dans game, ou alors au pire utiliser des attributs statiques aux classes. Joueur, Zombie, Projectile, Sphère de collision, Interface d'affichage et même Monde/Carte sont des classes potentielles.

- Vous devrez remettre l'UML final de votre projet.

- Classes: faites les déclarations dans le .h, et les définitions dans le cpp.

- Attention aux erreurs de linkage: ne pas laisser une classe déclarée non définie, définissez la méthode et faites le "return" minimum pour que la méthode compile.

- Maintenez la séparation du code principal dans main dans les sections classiques du développement de jeu : initiation / entrées / logique / affichage.

- Favorisez les constantes plutôt que les valeurs inscrites (#define), et déclarez-les statiques si elles portent une valeur universelle pour chaque instance de la classe.

- Documentez votre code à l'aide des sections suivantes : Summary / Params / Returns. GhostDoc est votre ami. Mettez des descriptions détaillées dans les sections Summary svp

- Il est recommandé d'utiliser le moins possible les pointeurs durant ce TP. Ils ne sont pas strictement interdits, mais si vous décidez tout de même de les utiliser, gérez correctement votre mémoire et gérez les erreurs qui pourraient survenir. UTILISEZ VLD.

## TESTS UNITAIRES

Vous allez devoir tester les collisions. Vous allez devoir tester toutes les méthodes qui déterminent si un objet entre en collision avec un autre (Zombie/Munition, Zombie/Tireur).

Vous pouvez au départ utiliser de simples sphères de collision avec les valeurs des sphères des différents acteurs du jeu. Si le tout fonctionne essayez par la suite de tester directement avec les classes.

Les cas à tester seront

- Clairement pas de collision
- Pas de collision à la limite (on se frôle mais ne se touche pas).
- Collision à la limite (On se touche à peine).
- Collision franche

Comme élément bonus (voir plus bas), chaque méthode avec retour devrait être testée. Ce sera le seul bonus qui peut passer outre le 80%

## ORDRE DES OBJECTIFS SUGGÉRÉ

Voici l'ordre des tâches que vous deviez respecter pour vous assurer d'un déroulement efficace. Les classes créées lors des exercices peuvent vous servir de départ (Exemples SFML, balle rebondissante, Pokémon 1 à 3, et même l'exercice optionnel Asteroid).

- Ayez d'abord un personnage à l'écran
- Le personnage peut de déplacer
- Le personnage peut tirer des projectiles de base
- Les zombies apparaissent directement à l'écran (apparition soudaine; un par seconde environ)
- Les zombies convergent vers le joueur.
- Le tir sur le zombie abat le dit zombie.
- Le zombie qui touche le joueur abat le joueur
- Testez vos collisions immédiatement (système de tir/collisions fonctionnel)
- Mécanique de vies du joueur et de ses respawns
- Ajoutez l'interface de jeu des points et des vies.
- Ajouter les icones des différents types de munitions (5% par Zombie abattu environ)
- Concrétisez les différentes armes spéciales.
- Ajoutez les munitions à l'interface de jeu.
- Implémentez votre UML final (S'il vous reste peu de temps faites-le un peu d'avance)
- Complétez la documentation de votre code.
- Implémentez les fonctionnalités boni éventuelles que vous voudriez ajouter.

## ASSETS GRAPHIQUES

Des assets graphiques vous seront proposés. Par contre, ils sont assez basiques. En attendant, rien ne vous empêche d'utiliser les vôtres ou d'utiliser des placeholders.

L'esthétique graphique ne sera pas évaluée (bien qu'une interface agressante puisse être pénalisée).

## BONUS

Pour qu'un élément bonus puisse s'appliquer un total de 80% de la note de base est nécessaire avant que les bonus soient pris en compte, et tous les éléments des objectifs suggérés doivent être faits. Faites une bonne base avant tout.

Voici quelques suggestions de bonus potentiels :

- Écran de départ et de fin avec navigation fonctionnelle.
- Animation de spawning du zombie, avec pivot et zoom comme dans le jeu original.
- Un zombie ne peut pas se superposer à un autre zombie; ils respectent entre eux leurs collisions.
- Différents types de zombies (ex : zombies plus gros qui demandent plus de munitions avant de mourir, zombies plus rapides, zombies qui se divisent, etc.)
- Si vous réussissez le bonus précédent, vous pourriez considérer que le joueur a des points de vie et que le contact avec le zombie fait perdre des points de vie. Mais ici toutes les collisions doivent être correctement respectées. Pour compenser la résistance du joueur, il faut... plus de zombies.
- Si les deux précédents bonus sont réussis : obstacles et murs dans le jeu (attention, très complexe; les zombies doivent pouvoir les contourner)
- Sauvegarde des meilleurs scores. Dans ce cas, l'écran final pourrait afficher les 10 meilleurs.
- Inventaire des munitions : Toutes les armes peuvent être accumulées et le joueur peut changer d'arme, mais le tout doit être correctement illustré dans l'interface. Dans ce cas, les bonus devraient donner moins de munitions.
- Système sonore (Musique et FX sonores)
- Tests unitaire complets (compte même sans le 80%)
- Toute autre idée qui peut améliorer le jeu. Consultez votre professeur avant de vous lancer.

## CONTEXTE DE RÉALISATION ET DÉMARCHE DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **individuellement**.

**Biens livrables** :

- Projet avec code source de l'application (code bien documenté)
- Fiche de fonctionnalités numérique.
- Diagramme UML

**Conditions de remise** :

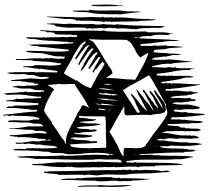
- L'application doit compiler et être fonctionnelle (On ne se battra pas longtemps pour faire compiler votre projet si ce n'est pas le cas au départ.)
- Remise via LEA

**Dates de remise** :

**Groupe 01 : 26 septembre en fin de journée**
**Groupe 02 : 27 septembre en fin de journée**

## 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** | |
|---|---|
| **Codification des classes (016T)** | |
| Critères d'évaluation | |
| - Utilisation correcte du C++. Code clair et bien documenté. Exploitation judicieuse des possibilités du langage. Algorithmes compréhensibles. Division du code en .h et .cpp fait correctement. | **15%** |
| - Modèle orienté objet correct. Utilisation optimale des classes. Respect de l'encapsulation. Modèle cohérent. Bonne utilisation des const. | **25%** |
| - Diagramme de classe UML de votre projet | **5%** |
| **Utilisation et amélioration du code existant (0176)**<br>**Développement d'une application dans un environnement graphique (017C)** | |
| Critères d'évaluation | |
| Partie fonctionnelle (orientation/direction du tireur/des zombies, élimination des zombies/du tireur, (ré) apparition du tireur, des zombies et des boni. Comportement correct des armes spéciales. Points et interface UI. | **35%** |
| Système de collision complet entre tous les acteurs. Comportement approprié des projectiles. | **10%** |
| Qualité générale du projet. | **5%** |
| Système de tests unitaires. | **5%** |
| **TOTAL** | **100%** |
| **Bonus discrétionnaires : ajout potentiel de 10% pour une note maximale de 100%** | |