

**420-W42-SF**  
**PROGRAMMATION DE JEUX VIDÉO I**  
**TRAVAIL PRATIQUE #1**  
**MARS INVASION**  
**PONDÉRATION : 30%**

**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 de la qualité et de documentation.

**MISE EN CONTEXTE ET MANDAT**

Ce travail pratique a pour but premier de créer un jeu complet en 2D, sur le principe du '*Twin Stick Shooter*' dans un environnement C++ / SFML.

**SPÉCIFICATIONS DE L'APPLICATION DE BASE**

Vous disposez d'un projet de départ. Dans ce projet, un personnage principal, un ennemi ainsi qu'un texte sont affichés comme références.

Il est possible de se déplacer dans un espace plus grand que l'écran. Cet espace constitue l'espace de jeu défini. Vous constaterez que seule la caméra se déplace. Le personnage principal et le héros ne bougent pas. Vous devrez faire en sorte que la caméra soit centrée sur le personnage principal.

Votre jeu doit respecter les spécifications suivantes :

**Le personnage principal**

- Le personnage peut se déplacer dans l'environnement de jeu.
- Il ne peut pas sortir des limites du jeu
- Le personnage joueur possède un nombre illimité de munitions qu'il peut tirer avec une certaine cadence (laissée à votre discrétion).
- Les ennemis sont tués dès qu'ils sont atteints par un projectile.
- Le joueur meurt s'il est touché par un ennemi et réapparaît au même endroit où il a été tué.
- Le joueur est invincible pendant 3 secondes après avoir été tué. Vous pouvez ajuster la durée de transparence sur le joueur pour illustrer cela.
- Le joueur qui a récupéré un bonus est affiché avec une teinte dorée
- Le joueur qui est en état de bonus ne meurt pas s'il est touché. Par contre son état de bonus prend fin et l'ennemi est neutralisé.

### Les ennemis

- Les ennemis apparaissent à un rythme constant laissé à votre discrétion.
- Un ennemi devrait apparaître à une distance minimale du joueur. Si vous le souhaitez, des ennemis peuvent apparaître hors de la vue, mais pas trop loin et jamais hors de la zone de jeu. Vous pouvez au début les faire apparaître directement, sans aucun "effet" pour garder les choses simples.
- Il y a trois textures pour les ennemis. Un ennemi créé reçoit une texture au hasard au moment de sa création.
- Les ennemis se dirigent constamment vers le joueur.
- Si le joueur est tué, les ennemis s'éloignent de l'endroit où le joueur est mort pendant environ deux secondes et se remettent à avancer vers le joueur ensuite.
- Un ennemi est tué s'il reçoit un projectile.
- Vous devez allouer au début de la partie les ennemis. Utilisez une méthode d'allocation fixe, et placez-y directement vos ennemis avant le début de la partie. Déterminez un état actif / inactif, mais l'ennemi doit toujours être en mémoire (pas de déchargement-rechargement)
- Le pointage obtenu par le joueur apparaît brièvement à l'endroit où l'ennemi a été tué.


### Les projectiles

- Il y a deux types de projectiles.
- Le type « normal » est disponible par défaut. Le type « spécial » est accessible si le joueur ramasse un bonus (*power-up*).
- Le projectile « spécial » n'est pas détruit au contact d'un ennemi et continue son déplacement.
- Les projectiles « spéciaux » sont accessibles après avoir récupéré un bonus (voir section suivante).
- Les projectiles doivent être pré-créés. Aucun projectile ne peut/doit être créé pendant le déroulement du jeu.
- Les projectiles qui sortent de la surface du jeu sont désactivés et retournent dans le *pool* de projectiles.
- Un son est émis lorsqu'un projectile est lancé.

### Les bonus

- Un bonus apparaît aléatoirement (ex. dans 20% des cas) lorsqu'un ennemi meurt.
- Un bonus apparaît à l'endroit où se trouvait l'ennemi.
- Un bonus peut être récupéré par le personnage principal.
- Les bonus doivent être pré-crées. Aucun bonus ne peut/doit être créé pendant le déroulement du jeu.
- Les bonus récupérés sont désactivés et retournent dans le *pool* de bonus.
- Un son est émis lorsqu'un bonus est récupéré.

### La partie

- Le monde est plus grand que la surface affichée.
- La caméra du jeu doit suivre le joueur à l'intérieur des limites du jeu. Elle doit aussi cesser de se déplacer lorsque le joueur arrive aux extrémités de la surface de jeu.
- Vous devez créer un système de pointage dans lequel un joueur obtient des points pour chaque ennemi tué et, si vous le souhaitez, pour chaque seconde où il reste en vie.
- Le joueur a cinq vies et en gagne une nouvelle par tranche de X points (X est laissé à votre discrétion mais assurez-vous qu'il soit atteignable assez facilement).
- Il est possible de faire pause dans le jeu à l'aide de la touche  ou « start » sur le contrôleur.
- La partie se termine si le joueur n'a plus de vie.
- Le pointage du joueur est affiché à l'écran en permanence tout comme le nombre de vies restantes.

Notez qu'il vous est recommandé d'expérimenter avec vos propres métriques de jeu (vitesse du personnage, des ennemis, vitesse des projectiles, etc.) mais vous pouvez demander à votre professeur les métriques qui furent utilisées pour un acteur si cela peut vous rassurer.

Vous pouvez référer à l'**exécutable fourni** pour avoir une idée du résultat à produire.

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.

## CONTRAINTES ET RECOMMANDATIONS DE DÉVELOPPEMENT

- Dans les méthodes, toujours passer les objets par référence, sauf si vous avez vraiment besoin d'une copie.
- Utilisation systématique de `const`, à la fin des méthodes qui ne modifie pas l'objet courant (dont les `get` et le `draw`) et `const` sur les paramètres en entrée que l'on ne veut pas modifier (donc sur la grande majorité des `set`).
- Essayez de ne pas faire une classe `Game` trop massive et donnez les responsabilités appropriées à chaque classe.
- L'utilisation d'un `ContentManager` est fortement essentielle. **NE CHARGER CHAQUE TEXTURE QU'UNE SEULE FOIS.**
- 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 codées en dur. Vous serez pénalisés pour chaque valeur inscrite directement dans le code.
- Faites parler votre code :
  - Favorisez un bon nommage de vos classes, attributs et méthodes.
  - Pas besoin de tout commenter.
- Portez attention à l'utilisation d'allocation dynamique. Si vous utilisez l'opérateur `new`, assurez-vous d'utiliser adéquatement l'opérateur `delete`. L'utilisation de VLD peut s'avérer essentielle.
- Assurez-vous de compiler **sans avertissement** tant en `Debug` qu'en `Release`. Vous serez pénalisés s'il reste des avertissements dans votre remise finale.

## 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)

### Conditions de remise :

- L'application doit compiler **sans avertissement** et être fonctionnelle.
- Remise via LEA dans un `.zip` duquel vous aurez retiré les dossiers `.vs`, `Debug`, `Release` et `extern` (si applicable)

**Date de remise : voir la date spécifiée sur LEA.**

## MODALITÉS D'ÉVALUATION

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

Dans l'éventualité où vous récupérez 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.

La **grille d'évaluation** utilisée pour ce travail est jointe à cet énoncé.