

420-V21-SF
PROGRAMMATION DE JEUX VIDÉO II
TRAVAIL PRATIQUE #3
GEOMETRY WARS
PONDÉRATION : 10%

OBJECTIFS PÉDAGOGIQUES

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

- Procéder à la codification d'une classe (compétence 016T-2).
- Valider le fonctionnement d'une classe (compétence 016T-4)
- Générer la version exécutable d'un programme (compétence 016T-5)

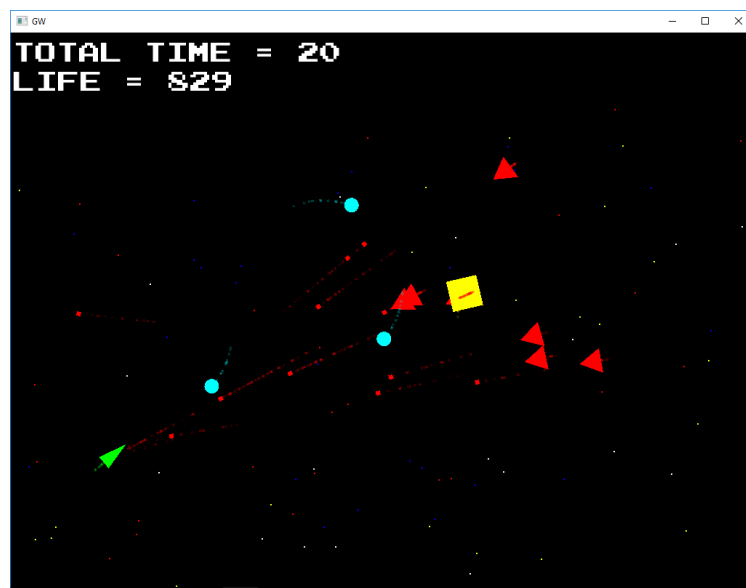
De même, il permet à l'étudiante ou à l'étudiant d'appliquer les standards de conception, de documentation et de programmation.

MISE EN CONTEXTE ET MANDAT

Pour ce dernier travail, vous devez réaliser une variante du jeu « Geometry Wars ». Dans le jeu publié en 2003, le joueur pilote un vaisseau spatial qui se déplace dans l'écran et qui peut tirer dans toutes les directions. Le joueur dispose d'un nombre restreint de bombes qui peuvent détruire tous les ennemis présents sur la surface du jeu.

Le principe de base du jeu est de survivre le plus longtemps possible. Vous pouvez vous référer à Youtube pour avoir une idée de la jouabilité de ce jeu.

L'idée n'est pas de reproduire fidèlement l'un d'eux mais plutôt d'appliquer les concepts vus en classe. La version originale du jeu comporte plus de fonctionnalités que celle demandée dans ce travail.













SPÉCIFICATIONS DE L'APPLICATION

Votre application doit notamment respecter les contraintes suivantes :

Fonctionnalités des étoiles :

- Le jeu doit afficher en tout temps une quantité à peu près fixe d'étoiles (au moins une centaine) en arrière-plan.
- Les étoiles doivent donner l'impression de se déplacer en fonction de la direction dans laquelle le héros se déplace. Elles doivent donc se déplacer en sens inverse du héros pour créer un effet de mouvement.

Fonctionnalités du joueur :

- A chaque tour, le joueur doit pouvoir déplacer son personnage à l'aide des touches de votre choix (    ou    ) et lancer des projectiles standards à l'aide de la touche  (« espace »).
- Le joueur ne peut pas sortir de la surface de jeu.
- Le joueur peut lancer des projectiles pour détruire les ennemis. Il doit s'écouler un certain délai entre chaque lancement de projectile.
- Le joueur dispose d'un certain nombre de points de vie au démarrage de la partie (c'est vous qui devez le déterminer). Il perd des points lorsqu'il entre en contact avec un ennemi ou reçoit des projectiles de la part des ennemis. Le nombre de points de vie doit être affiché à l'écran en tout temps d'une manière claire
- Le joueur doit pouvoir lancer des bombes qui détruiront tous les ennemis sur la surface du jeu. Il dispose d'un nombre restreint de telles bombes au début de la partie (c'est vous qui devez le déterminer). Le joueur peut lancer ses bombes en appuyant sur la touche . Il n'est pas nécessaire d'avoir une rétroaction visuelle à l'écran. Vous pouvez uniquement détruire tous les ennemis présents sans faire afficher de bombe en tant que telle.

Fonctionnalités des ennemis :

- Il doit y avoir au moins 3 types d'ennemis : les triangles, les carrés et les cercles.
- Chaque type d'ennemi doit avoir son comportement particulier. Vous êtes libres de choisir le comportement que vous préférez pour chaque type d'ennemi (ex. déplacement aléatoire, attaque agressive, etc). Vous devrez détailler dans le document de remise les comportements en question.
- En tout temps, il doit y avoir au moins 5 ennemis de types carrés ou cercles en même temps sur la surface de jeu. Il peut y avoir un nombre quelconque de triangles.
- Les ennemis ne doivent pas quitter la surface du jeu.
- Lorsqu'ils sont détruits, les cercles et les carrés doivent exploser et générer des triangles qui attaqueront à leur tour le héros.
- Un ennemi est détruit lorsqu'un projectile du héros l'atteint.

- Lorsqu'un ennemi entre en contact avec le joueur, ce dernier perd des points de vie et l'ennemi est détruit.

Fonctionnalités des projectiles :

- Lorsqu'un projectile sort de la surface de jeu, il doit être retiré de la « partie ».
- Les ennemis ne doivent pas pouvoir se détruire entre eux (i.e. les tirs « amis » n'endommagent pas).

Fonctionnalités du système de localisation :

- Le jeu doit proposer un système de localisation. Le système de localisation demandé réfère au fait que deux langues doivent être supportées par le jeu (français et anglais).
- Le système de localisation doit être implémenté à l'aide d'un **dictionnaire**.
- Les chaînes de caractères requises pour les langues doivent être lues dans un fichier texte dans le format suivant :



ID_TOTAL_TIME==>Temps total---Total time

où ID_TOTAL_TIME représente l'identifiant à utiliser dans le jeu pour récupérer la chaîne de caractères appropriée, Temps total représente la valeur requise en français et Total time celle en anglais.

- Le système de localisation doit être implémenté à l'aide d'une classe `StringTable` implémentée en *singleton*.
- Par exemple, pour afficher le pointage du joueur dans le jeu dans une étiquette de texte nommée `timeText`, l'instruction requise serait équivalente à celle-ci :

```
timeText.DisplayedString=StringTable.GetInstance().GetValue(CurrentLanguage,
"ID_TOTAL_TIME" ) + ":" + totalTime.ToString( );
```

où `CurrentLanguage` désigne, comme vous vous en doutez, la langue d'affichage présentement sélectionnée et `ID_TOTAL_TIME` représente l'identifiant associé à la chaîne de caractères qui nous intéresse.

- Il doit être possible pendant le jeu de changer la langue d'affichage avec  (français) ou  (anglais).

Note. Le jeu demandé ne gère que 2 identifiants. Si vous ne parvenez pas à créer la classe `StringTable` au complet, vous pouvez malgré tout en faire une implémentation avec des `if / else if` qui fonctionnera pour les deux chaînes nécessaires.

Vous pouvez aussi écrire les tests unitaires et ce, même si le chargement ne fonctionne pas. Ainsi, vous pouvez obtenir une bonne partie des points sans pour autant que tout soit fonctionnel.

Consignes globales

- Le jeu doit être programmé en utilisant les standards de programmation qui vous ont été communiqués en classe en début de session. En particulier, le code doit :
 - Être correctement indenté.
 - Ne doit contenir qu'une seule instruction par ligne.
 - Utiliser les structures de contrôles adéquates (*while*, *for*, *if*, etc).
 - Être séparé adéquatement en classes et méthodes pour faciliter la lisibilité et la réutilisation.
 - Utiliser les principes de programmation objet vus en classe notamment les principes d'héritage et de polymorphisme.

Vous devez également commenter à profusion les instructions, les déclarations de variables et les entêtes de méthodes.

- Vous devez aussi utiliser les propriétés C# (*get/set*) aussi souvent que cela est possible et pertinent. N'oubliez pas de porter attention à la visibilité des *get/set* ainsi qu'à la possibilité qu'une propriété C# ne soit accessible qu'en lecture seulement.
- Vous devez finalement utiliser et lever des exceptions lorsque cela est opportun. Soyez vigilants en ce qui a trait à la validité des paramètres reçus (ex. index d'accès à un tableau incorrect).
- Finalement, **vous devez produire tous les tests pertinents à la classe `StringTable`**

CONTEXTE DE RÉALISATION ET DÉMARCHÉ DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **seul**. Les pages qui suivent résument les étapes du projet ainsi que les biens livrables devant être remis à la fin de chacune d'elles.

Étape 1: Analyse de la situation et prise de connaissance des mandats

Dans cette étape, vous devez prendre connaissance de la demande et clarifier les éléments nécessaires avec votre professeur.

Biens livrables :

- Aucun.

Étape 2: Programmation de l'application

Vous devez procéder à la codification des classes contenues dans le diagramme de classes fourni (voir la section plus loin). Ce diagramme est partiel : il ne contient pas toutes les propriétés (*variables membres*) et plusieurs méthodes sont masquées. **Vous pouvez et devez donc ajouter les méthodes manquantes selon vos besoins.**

Vous devez également procéder à la codification des tests unitaires de la classe `StringTable`.

Pour cela, vous devez récupérer la base de code fournie sur LÉA.

Commencez par prendre connaissance du code fourni.

Complétez le code des deux projets (TP3 et TP3Tests).

Notez qu'à des fins de correction, vous pourrez être convoqués à une rencontre visant à expliquer les parties de code que vous avez réalisées. Vous devez donc être en mesure de bien comprendre toutes les parties (fonctionnalités, code, etc).

Biens livrables :

- Code source de l'application documenté (selon les standards établis dans le cours) et code **documenté** des tests unitaires que vous coderez.
- Pour les tests, créez les différents tests requis (voir la grille de correction) et n'oubliez pas de documenter adéquatement chaque test pour décrire son rôle.

Méthode:

- Vous devez remettre votre code source dans **une archive .zip** identifiée en fonction de votre nom. Par exemple, pour le travail remis par Pierre Poulin, l'archive devrait être nommée TP3_PPoulin.zip.

Cette archive doit être déposée sur LÉA **avant le 14 mai 2017 à l'heure indiquée sur LÉA (avant minuit).**

N'oubliez pas de documenter adéquatement vos entêtes de fichiers.

MODALITÉS D'ÉVALUATION

Vous trouverez sur LÉA la grille de correction qui sera utilisée pour le travail. **Cette grille indique la pondération accordée à chacune des parties du projet.**



Ce travail peut être bonifié largement par celle et ceux qui le souhaitent.

Notez qu'afin d'**encourager les initiatives et améliorations personnelles**, un facteur multiplicatif sera appliqué à la toute fin permettant ainsi de bonifier les travaux pour lesquels il y a eu une nette amélioration par rapport à la demande.

- Tous les **biens livrables** de chacune des étapes devront être **remis à temps** et selon les modalités spécifiées.
- Nous rappelons que la **présence à tous les cours (de corps et d'esprit)** est **FORTEMENT RECOMMANDÉE**.
- Je vous recommande aussi de profiter de toutes les périodes de disponibilité qui vous sont offertes pendant la semaine.

STRATÉGIE DE DÉVELOPPEMENT.

Afin d'éviter de devoir gérer beaucoup de bogues en même temps, je vous suggère fortement de procéder à la codification de manière structurée.

Plusieurs des classes à coder et des interactions entre elles sont similaires à certaines classes que vous avez codées en exercices pendant la session. **Vous pouvez évidemment vous en inspirer.**

Je vous suggère d'y aller dans l'ordre suivant (sans respect pour les mandats) :

- i. Classe `Hero` : affichage, déplacements, gestion des munitions, etc. Avant d'aller plus loin, assurez-vous que votre joueur est capable de se déplacer dans l'environnement.
- ii. Classe `Projectile` : affichage, déplacements, retrait si sortis de la surface de jeu, etc. Avant d'aller plus loin, assurez-vous que cela est fonctionnel.
- iii. Classe `Enemy` (et ses dérivées) : affichage, positionnement au début de la partie, collision avec les projectiles, etc.
- iv. Gestion de la partie : nombre de points de vie du joueur, affichage temps et des points de vie dans l'interface et détection de la fin de partie.

À n'importe quel moment, vous pouvez réaliser la classe `StringTable`.

