

**420-V21-SF**  
**PROGRAMMATION DE JEUX VIDÉO II**  
**TRAVAIL PRATIQUE #3**  
**MILLIPÈDE**  
**PONDÉRATION : 6%**

**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 connu dans les années 80 comme « Centipede ». Dans le jeu original, un joueur contrôle un nain qui doit protéger son jardin des insectes. Le principe de base est d'accumuler le plus de points possible en détruisant les opposants (serpents ou vers et araignées). Vous pouvez vous référer à l'adresse <https://www.youtube.com/watch?v=xGEZ3NNH6cs> pour avoir une idée de la jouabilité de ce jeu.


Il existe une très grande variété de jeux similaires reprenant les mêmes fonctionnalités. L'idée n'est pas de reproduire fidèlement l'un d'eux mais plutôt d'appliquer les concepts vus en classe.




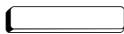
## SPÉCIFICATIONS DE L'APPLICATION

Votre application doit notamment respecter les contraintes suivantes :

### Fonctionnalités globales :

- Elle doit afficher au minimum un personnage contrôlé par le joueur, un « serpent », des champignons et une ou plusieurs araignées.
- Le jeu doit permettre de quitter la partie à l'aide du bouton habituel  sans toutefois permettre le redimensionnement de la fenêtre. Notez que la fenêtre doit être de dimensions correctes au démarrage du jeu.
- Lorsqu'une partie est terminée normalement (une boîte de dialogue doit demander au joueur s'il souhaite continuer ou quitter le jeu.

### Fonctionnalités du joueur :

- A chaque tour, le joueur doit pouvoir déplacer son personnage à l'aide des touches  et faire feu à l'aide de la touche  (« espace »).
- Le joueur ne peut pas sortir de la surface de jeu.
- Vous devez modifier le fichier `player.bmp` pour une image plus esthétique.
- Le joueur peut lancer des projectiles pour détruire le(s) serpent(s), la (les) araignée(s) et les champignons. Il doit s'écouler au moins 500ms entre chaque lancement de projectile.
- Le joueur dispose de 3 « vies » au démarrage de la partie. Il perd une vie lorsqu'il entre en contact avec un serpent ou une araignée. Le nombre de vies doit être affiché à l'écran en tout temps.

### Fonctionnalités des araignées :

- Une araignée est ajoutée à la surface de jeu de manière aléatoire. Il peut y avoir plusieurs araignées en même temps. Les paramètres d'ajout (fréquence, conditions, etc) des araignées sont laissés à votre discrétion.
- Les araignées se déplacent de manière aléatoire mais, « de temps en temps », elles décident de foncer sur le joueur. La notion de « de temps en temps » est laissée à votre discrétion.
- L'araignée est détruite lorsqu'un projectile l'atteint.
- Lorsqu'une araignée est détruite, elle se change en champignon.
- Lorsqu'une araignée entre en contact avec le joueur, ce dernier perd une vie et tous les adversaires présents sur la surface de jeu se transforment en champignons.

### Fonctionnalités des serpents :

- Il y a toujours au moins un serpent qui se déplace. Si tous les serpents ont été détruits, un autre doit apparaître en haut à gauche de l'écran.
- Un serpent est coupé en 2 lorsqu'un tir du joueur l'atteint en son milieu. Si le tir frappe l'une des extrémités, le résultat attendu est que la partie atteinte (la tête ou la queue) soit retirée du serpent en question et que le reste continue à se déplacer. Si un serpent est coupé en deux, la première partie doit continuer dans la direction où elle se dirigeait avant la réception du tir tandis que l'autre doit changer de direction. La partie du serpent où a été reçu le tir doit se changer en champignon.
- Lorsqu'il rencontre un champignon, chaque serpent « descend » dans la surface de jeu et change de direction. Si jamais un champignon est présent immédiatement sous la tête du serpent, le serpent « passe par-dessus » ce champignon (il ne reste pas bloqué).
- Lorsque la tête du serpent sort de la surface du jeu par le bas, elle doit être retirée du serpent. Le serpent ne doit pas alors changer de direction.
- Lorsqu'un serpent entre en contact avec le joueur, ce dernier perd une vie et tous les adversaires présents sur la surface de jeu se transforment en champignons.

### Fonctionnalités des champignons :

- Au démarrage du jeu, la surface de jeu doit contenir un nombre variable (aléatoire) de champignons dont les positions auront été déterminées de manière aléatoire. Aucun des champignons ne doit être superposé avec un autre.

### Fonctionnalités des munitions :

- Un joueur possède un nombre maximal de munitions. Au début, ce nombre est fixé à 20. Lorsque ce nombre descend en bas de 5, un « power-up » est affiché à l'écran. Le joueur peut alors le récupérer et obtenir 20 munitions additionnelles.

### Fonctionnalités du leaderboard :

- Le jeu doit proposer un système de pointage. Ce système est laissé à votre guise mais doit faire en sorte qu'à la fin de toute partie, le joueur obtient un pointage (valeur entière). Ce pointage doit en tout temps être affiché à l'écran
- Lorsqu'une partie se termine, un leaderboard est affiché au joueur. Ce leaderboard doit contenir les 10 meilleurs pointages réalisés dans le jeu.
- Le leaderboard doit être persistant, i.-e. il doit être sauvegardé (et relu!) dans un fichier texte. Le format du fichier doit être :

```
pointage1,nom du joueur1;  
pointage2,nom du joueur2;
```

[...]

Pointage10,nom du joueur10;

- Lorsqu'un joueur réalise un pointage qui lui vaut une place sur le leaderboard, le jeu doit lui demander d'inscrire son nom. Son nom et son pointage sont alors ajoutés au leaderboard. Vous devrez donc créer un formulaire approprié permettant de saisir et valider le nom du joueur.
- L'interface du leaderboard et celle permettant de saisir le nom du joueur doivent être soignées. Portez une attention à l'esthétisme.
- Lorsqu'une entrée est faite au leaderboard, ce dernier doit obligatoirement être sauvegardé.

#### Fonctionnalités du système de journalisation :

- Le jeu doit proposer un système de journalisation.

Chaque événement « important » doit être signalé dans un journal (dans un *log*) :

- Démarrage du jeu
- Début d'une partie
- Fin d'une partie
- Ajout d'une nouvelle entrée dans le leaderboard
- Etc.

Notez le format de la date et de l'heure

Le format du fichier de journalisation est le suivant :

```
2015-05-02 21:45:48: Début d'une nouvelle partie
2015-05-02 21:45:58: Fin de la partie
2015-05-02 21:46:03: Nouvelle entrée au leaderboard: 7600-PKSubban
```

- Le système de log doit être implémenté à l'aide d'une classe *Logger* implémentée en *singleton*.

#### 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 et le principe de récursivité en ce qui a trait au déplacement des fantômes.

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, fichier introuvable, etc).
- Finalement, vous devez produire tous les tests pertinents à la classe `Leaderboard`.

## CONTEXTE DE RÉALISATION ET DÉMARCHÉ DE DÉVELOPPEMENT

Ce travail pratique doit être réalisé **en équipe de 2 en fonction des mandats qui sont présentés ci-après**. 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 répartir entre les deux coéquipiers les deux mandats qui vous sont confiés. Veuillez noter que **la correction sera faite par mandat**, i.-e. les deux coéquipiers n'auront pas nécessairement la même note.

<u>Mandat 1</u>	<u>Mandat 2</u>
<u>Structures dynamiques</u>	<u>Structures dynamiques</u>
- Gestion des parties du serpent et des multiples serpents lorsque séparés par les tirs du joueur.	- Fonctionnalités des champignons : ajout, destruction, génération aléatoire au début du jeu - Gestion des tirs: Destruction d'un champignon atteint par un tir du joueur.
<u>Fichiers textes</u>	<u>Fichiers textes</u>
- Fonctionnalités du système de journalisation des événements ( <i>log</i> )	- Gestion du <i>leaderboard</i> (sauvegarde + lecture)
<u>Jouabilité</u>	<u>Jouabilité</u>
- Fonctionnalités des serpents : génération, déplacements, séparations et suppressions des serpents lorsque ces derniers quittent la surface du jeu par le bas.	- Fonctionnalités du joueur : déplacement, affichage, tir des projectiles, etc.
- Gestion de la fin de partie et création de l'interface permettant de saisir le nom du joueur qui a réalisé un meilleur pointage.	- Gestion du pointage : pendant la partie
- Gestion des vies du joueur	- Fonctionnalités du <i>leaderboard</i> : création, lecture, sauvegarde,

	affichage, saisie des noms associés aux nouveaux meilleurs pointages, ajouts.
- Fonctionnalités des munitions du joueur	Fonctionnalités des araignées : génération, déplacements, affichage, etc.

#### **Mandat commun aux deux coéquipiers**

- Tests du leaderboard
- Fonctionnalités globales du jeu : fermeture de l'application, dimensions de la fenêtre, confirmation de la fin du jeu.

#### **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*)

Vous devez également procéder à la codification des tests unitaires pertinents pour le leaderboard.

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 TestLeaderboard) selon les mandats attribués.

**Attention.** Puisque la correction est individuelle, **chaque section de code que vous réaliserez devra être clairement identifiée.** Par exemple, si j'ajoute la méthode `GetSize` à une classe, il faut que j'inscrive le code suivant.

```

// <ppoulin>
public Size GetSize()
{
    return [...]
}
// </ppoulin>
```

Identification claire de la section de code produite

#### **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, utilisez la « coquille » du projet fourni et créez les différents tests requis. Documentez adéquatement chaque test pour décrire son rôle.

## Méthode:

- Vous devez remettre votre code source dans une archive .zip identifiée en fonction des noms des deux coéquipiers. Par exemple, pour l'équipe formée de Pierre Poulin et François Paradis, l'archive devrait être nommée TP2\_PPoulinFParadis.zip. Cette archive doit être déposée sur LÉA **avant le 20 mai 2015 à 23h59**.

**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.**

- 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 `Player` : affichage, déplacements, gestion des munitions, etc. Avant d'aller plus loin, assurez-vous que votre joueur est capable de se déplacer dans l'environnement sans dépasser les frontières du jeu.
- 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 `Mushroom` : affichage, positionnement aléatoire au début de la partie, collision avec les projectiles, etc.
- iv. Classe `Spider` : affichage, génération aléatoire, collision avec les projectiles, ajout d'un champignon si destruction, déplacements (aléatoire + ciblé vers le joueur), collision avec le joueur.
- v. Gestion de la fin de partie : nombre de vies du joueur, affichage du nombre de vies dans l'interface et détection de la fin de partie.
- vi. Gestion du pointage : ajout de points lorsqu'un adversaire est détruit, affichage du pointage dans l'interface.
- vii. `InputNameForm` : création de l'interface, saisie du nom du joueur (attention aux restrictions), affichage du formulaire.

- viii. Leaderboard : création de l'interface, gestion interne des pointages, chargement des pointages à partir d'un fichier, ajout d'un pointage dans les 10 premiers, sauvegarde du leaderboard, tests dans le projet de tests.
- ix. Classe `Snake` : génération, affichage, déplacements, collision avec les champignons, collisions avec le joueur, collision avec les projectiles, séparation en deux serpents lorsqu'atteint par un projectile, « retrait » lorsque le serpent atteint le bas de l'écran.

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



## DIAGRAMME DE CLASSES ET EXPLICATIONS

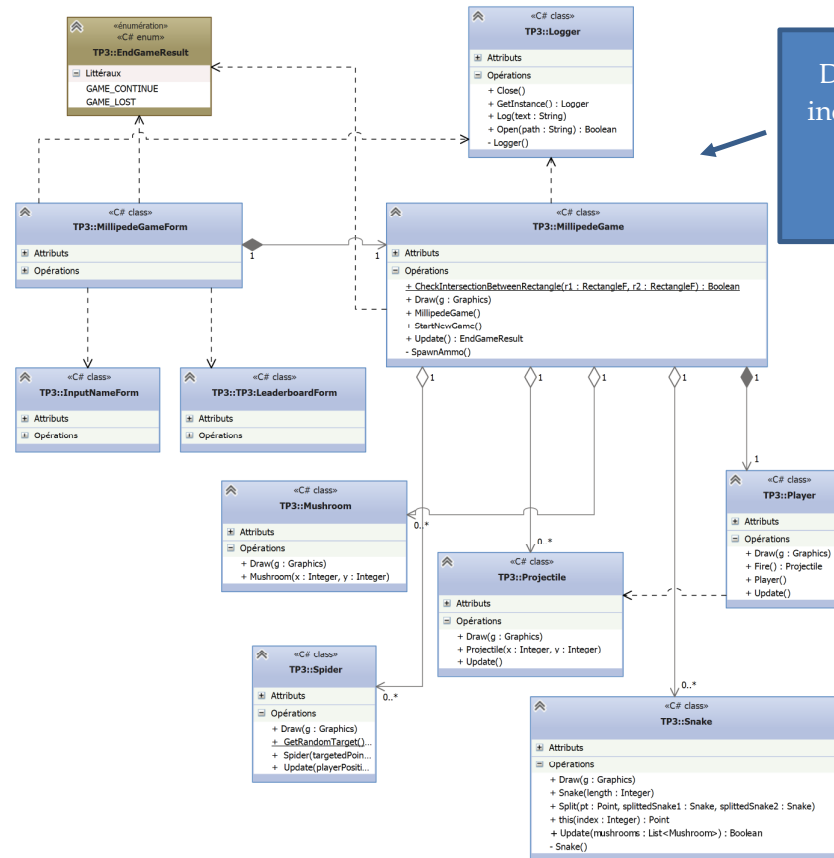


Diagramme de classes présenté à titre indicatif seulement. Il se peut que vous ayez à ajouter / modifier certaines méthodes.