

420-V21-SF
PROGRAMMATION DE JEUX VIDÉO II
TRAVAIL PRATIQUE #2
PACMAN
PONDÉRATION : 10%

OBJECTIFS PÉDAGOGIQUES

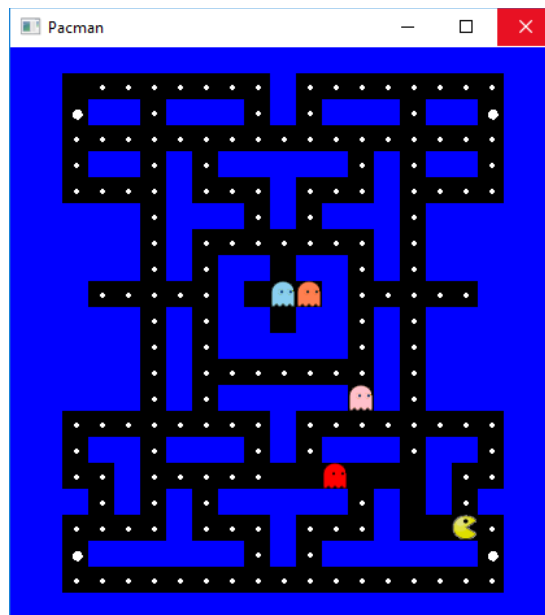
Ce travail vise à familiariser l'étudiante ou l'étudiant avec les objectifs suivants :

- 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)
- Mettre au point l'algorithme. (compétence 016W-2)
- Valider l'algorithme. (compétence 016W-3)

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 second travail, vous devez réaliser le jeu bien connu « Pacman ». Dans ce jeu, un joueur contrôle un « pacman » qui peut se déplacer dans un labyrinthe 2D afin de récupérer toutes les pastilles présentes dans le tableau de jeu sans se faire attraper par des fantômes dans le 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 :

- Elle doit afficher au minimum un pacman, quatre fantômes, des murs, des pastilles et des super pastilles.
- La configuration initiale du tableau de jeu doit être lue dans un fichier texte dans lequel les éléments devant se retrouver sur le tableau de jeu seront représentés par des nombres entre 1 et 6 sous la forme suivante :

1,1,1,1,1,4,1,4,1,1,1,2,1,1,4,1,6,1,1,1,1,1;



- 0→Rien, 1→ mur, 2→Fantôme, 3→PacMan, 4→Pastille, 5→ Super pastille, 6→Endroit où les fantômes vont se réfugier lorsqu'une super pastille est prise

- Une virgule sépare chaque nombre

- Un point-virgule termine chaque ligne sauf la dernière

Si le format de fichier est incorrect ou si le fichier est introuvable, un message doit être affiché à l'écran et le jeu doit se terminer.

Le format d'un fichier est correct si :

- a) Le tableau possède 22 lignes et 21 colonnes
 - b) Le tableau possède une et une seule cage à fantômes (un seul « 6 »).
 - c) Le tableau possède un et un seul pacman (un seul « 3 »).
 - d) Le fichier ne comporte pas de caractère inattendu.
- Le jeu doit permettre de quitter la partie à l'aide du bouton habituel .
 - A chaque tour, le joueur doit pouvoir déplacer le pacman à l'aide des touches .
 - Le jeu doit compter 4 fantômes qui tenteront d'attraper le pacman.
 - Les fantômes doivent être activés (i.e. commencer à bouger) avec un décalage d'environ 1 seconde. .
 - Chaque fantôme doit, à chaque tour, trouver le chemin le plus court dans le tableau lui permettant de se rapprocher du pacman. Pour cela, **un algorithme récursif doit obligatoirement être utilisé (voir plus loin)**.
 - Chaque niveau doit comporter un certain nombre de super pastilles qui, pendant le délai de votre choix (*ex. 5 secondes*), vont rendre le pacman invincible.
 - La partie se termine lorsque le pacman a terminé de récupérer les pastilles associées au niveau.
 - 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**.
- **Utiliser au moins 5 propriétés C#** de manière correcte, i.-e. pour des contextes qui le justifient.

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

- Finalement, vous devez **produire tous les tests unitaires** pertinents demandés.

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>Section algorithmique</u>	<u>Section algorithmique</u>
- Calcul des coûts (distances) entre un point et une destination	- Détermination du premier déplacement à effectuer à partir d'un point pour se rendre à destination
<u>Section programmation</u>	<u>Section programmation</u>
- Classe Pacman	- Classe Ghost
- Code de l'algorithme élaboré	- Code de l'algorithme élaboré
- Tests de l'algorithme élaboré	- Tests de l'algorithme élaboré
- Gestion de la fin de partie	- Gestion des super pastilles
- Gestion des pastilles	-

Mandat commun aux deux coéquipiers

- Classe `PathFinder` (Code général de la classe. Le code spécifique est par mandat)
- Classe `Grid` (Chargement des niveaux de jeu)
- Classe `PacmanGame` (Excluant les parties spécifiques aux mandats)
- Lecture des fichiers de niveaux.
- Fenêtre principale, interface utilisateur, types énumérés

Biens livrables :

- Aucun.

Méthode:

- s/o

Étape 2: Réalisation du volet algorithmique du mandat choisi

Vous devez procéder à l'élaboration de l'algorithme associé à votre mandat selon les consignes suivantes :

Mandat 1. Calcul des coûts (distances) entre un point et une destination

L'idée de l'algorithme est de calculer le nombre de déplacements minimal pour parcourir une grille 2D entre un point et une destination. Les tableaux présents à l'annexe 1 illustrent le comportement de l'algorithme. **Vous devez obligatoirement utiliser cet algorithme récursif pour solutionner cette partie du travail.** Aussi, vous devez **respecter la signature** (le prototype) des méthodes pour que les tests unitaires puissent s'exécuter.

Mandat 2. Détermination du premier déplacement à effectuer à partir d'un point pour se rendre à destination

L'idée de l'algorithme est d'utiliser le résultat de l'algorithme précédent et de partir de la destination pour retrouver le chemin jusqu'au point de départ. L'annexe 2 illustre l'algorithme en question. **Vous devez obligatoirement utiliser cet algorithme récursif pour solutionner cette partie du travail.** Aussi, vous devez **respecter la signature** (le prototype) des méthodes pour que les tests unitaires puissent s'exécuter.

Biens livrables (pour les deux mandats) :

- Algorithme décrit correctement et commenté sous la forme d'un document Word fourni sur LÉA.

Méthode:

- Vous devez remettre votre document Word identifié en fonction de votre nom et du mandat choisi (1 ou 2). Par exemple, pour l'étudiant nommé Pierre Poulin, le document devrait être nommé MandatX_PierrePoulin.zip. Assurez-vous de le remettre dans le bon travail sur LÉA.
- Cette archive doit être déposée sur LÉA **avant le 24 mars 2017 à 7h59**

Étape 3: Programmation de l'application

Vous devez procéder à la codification des classes contenues dans le diagramme de classes fourni (voir la section plus loin). Vous devez également procéder à la codification des tests unitaires pertinents.

Pour cela, vous devez récupérer la base de code fournie sur LÉA. **Cette base de code sera disponible à compter de vendredi le 24 mars 2017.**

Commencez par prendre connaissance du code fourni.

Complétez le code des deux projets (TP2ETU et TP2Tests) 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 [...];
}
```

Identification claire de la section de code produite

```
// </ppoulin>
```

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. Vous disposez d'une description des tests à produire. **Vous devez écrire le code correspondant aux descriptions fournies.**
- Vous disposez également d'une série de tests fournis mais placés en commentaires. Utilisez-les pour valider **au fur et à mesure** le code que vous écrirez.

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 2 avril 2017 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.

Je vous suggère d'y aller dans l'ordre suivant :

En dehors des heures de cours :

- a. Codification des éléments reliés à l'interface utilisateur : formulaire, gestion des touches, etc et des types énumérés.
- b. Lecture du fichier de niveau et génération de la grille de jeu (classe `Grid`). La lecture du fichier de niveau ressemble étrangement à la lecture des niveaux du travail pratique #1. Inspirez-vous en...
- c. Affichage des « pastilles » aux endroits appropriés.
- d. Codification des classes `Pacman` et `Ghost` (selon les mandats).

- e. Déplacement du pacman et d'un fantôme (selon les mandats) (avec les touches du clavier).
- f. Gestion de la fin de partie.

Pendant les heures de cours :

- i. Classe `PathFinder`. Codez l'algorithme associé à votre mandat.
- ii. Écriture des tests unitaires demandés.
- iii. Intégration du code de déplacement des fantômes dans le jeu.

DIAGRAMME DE CLASSES ET EXPLICATIONS

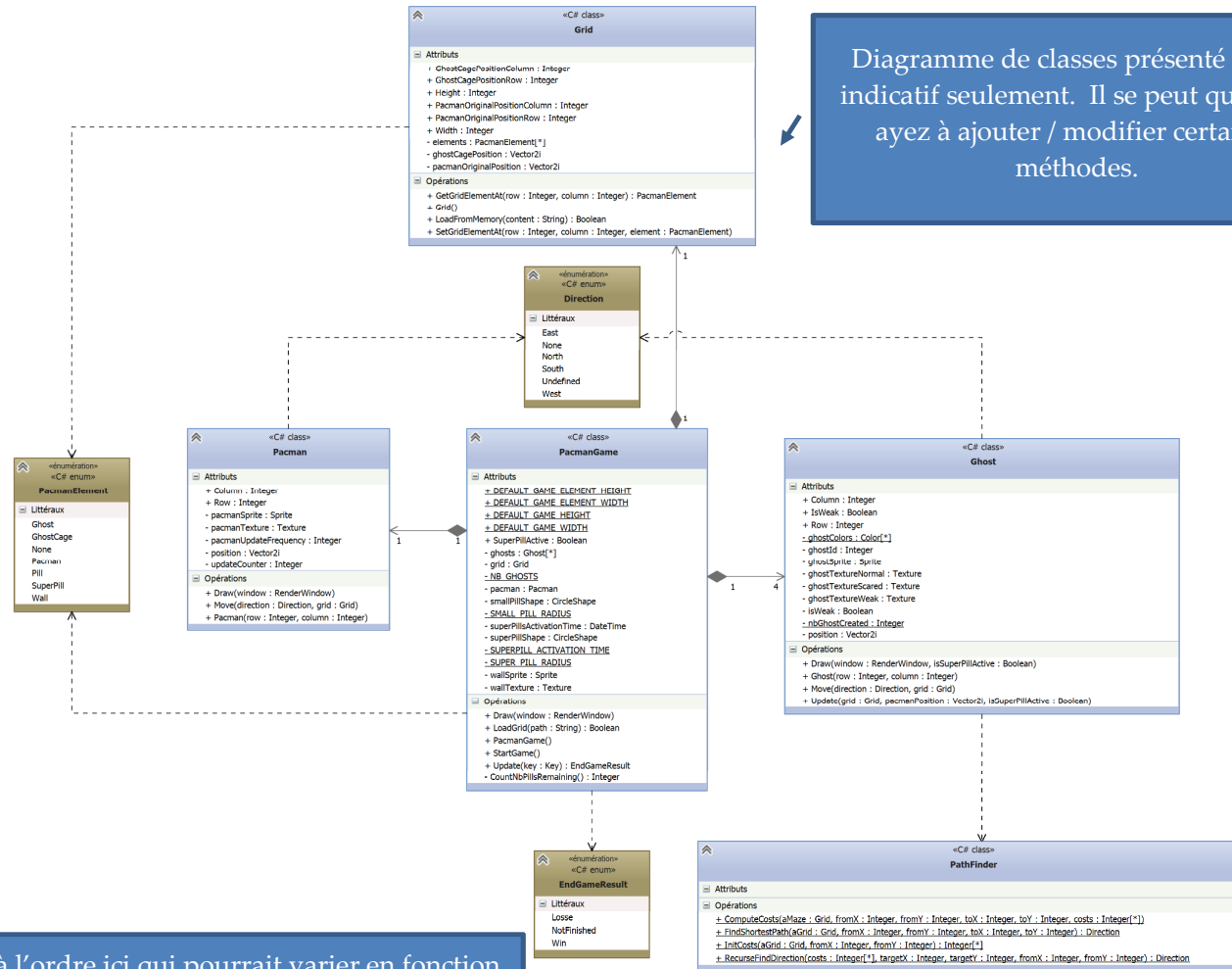
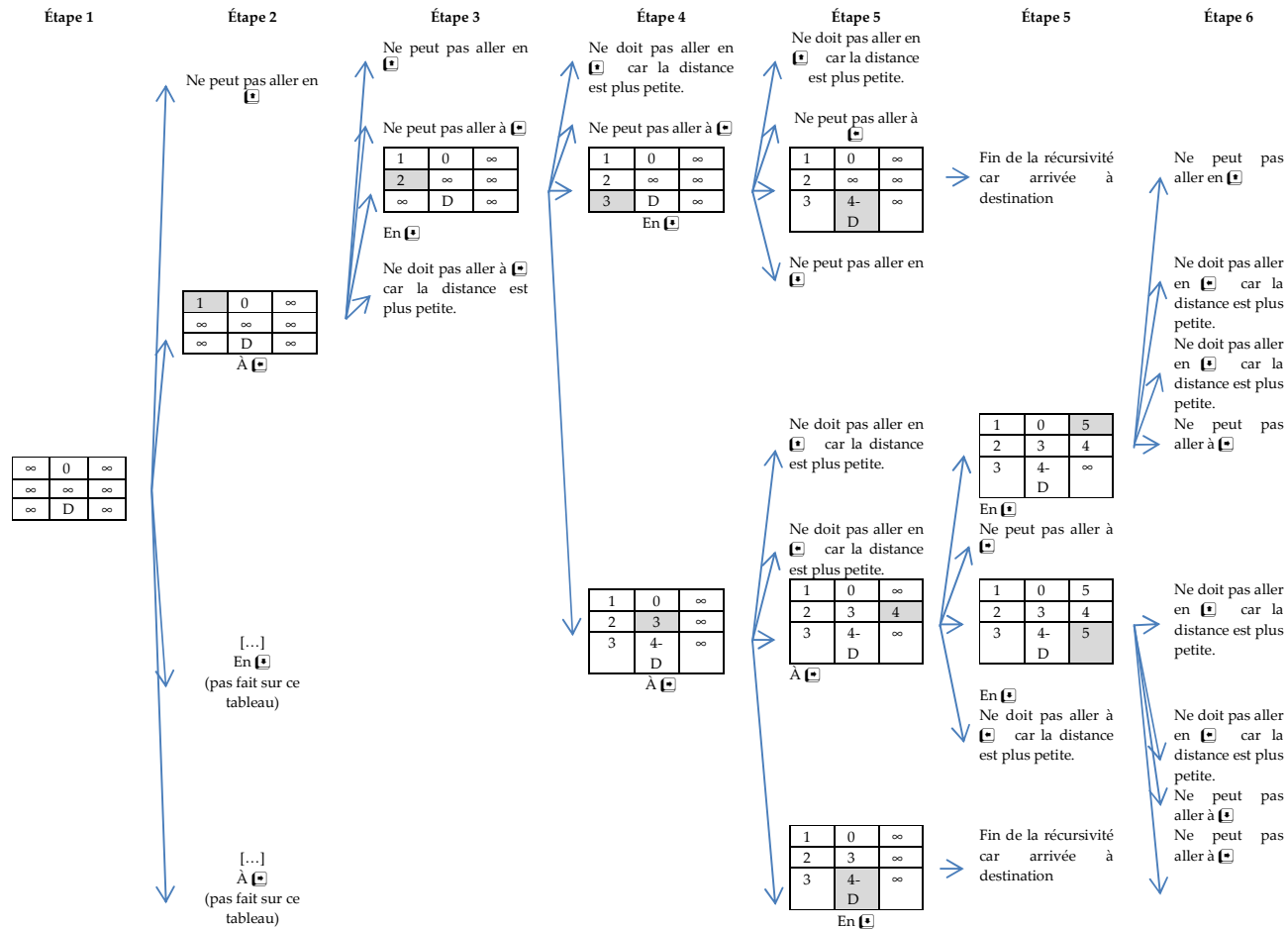


Diagramme de classes présenté à titre indicatif seulement. Il se peut que vous ayez à ajouter / modifier certaines méthodes.

Attention à l'ordre ici qui pourrait varier en fonction des spécifications fournies dans ce travail.

Annexe 1





Comportement de l'algorithme récursif du calcul des distances entre un point et une destination



À la toute fin, après l'exécution de tous les appels récursifs, le contenu final du tableau serait

1	0	1
2	1	2
3	2	3



La valeur associée à chaque case représente le nombre de déplacements , ,  et  requis entre l'origine (représentée par 0) et la destination.

Dans le travail pratique, il ne sera pas toujours possible d'aller dans la case voisine. Par exemple, si cette dernière est un mur, la distance devra toujours demeurer infinie (∞) ou `int.MaxValue` en C#.

Note importante.

Le tableau contenant les distances est partagé par tous les appels récursifs (c'est un type par référence). C'est cela qui nous permet de diminuer la taille du problème à chaque appel.

Annexe 2





Comportement de l'algorithme récursif de la détermination du chemin le plus court entre un point et une destination

À partir du résultat de l'algorithme précédent, il est possible de trouver le déplacement à réaliser à partir d'un algorithme récursif. Comment? En partant de la destination et en tentant de revenir à l'origine le plus rapidement possible, i.e. en parcourant les cellules dont la distance par rapport à l'origine est la plus petite.

Dans l'exemple précédent,

1	0	1
2	1	2
3	2	3



la distance totale entre la destination et l'origine est 2. Il suffit donc de regarder autour de la destination et de trouver les cases dont la valeur est 1 (→ 2-1). On sait ainsi que l'on s'approche de l'origine le plus rapidement possible. Il suffit de faire cela de manière récursive, pour toutes les cases , ,  et  dont la valeur est inférieure « de 1 » à celle de la destination jusqu'à ce que l'on ait atteint l'origine.

1	0	1
2	1	2
3	2-D	3

→

1	0	1
2	1	2
3	2	3

→

1	0	1
2	1	2
3	2	3

Supposons que l'on ait un exemple plus complexe,

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

Quand on retrouve l'origine, le point duquel on vient constitue le point vers lequel on doit aller pour rejoindre la destination le plus rapidement possible. Il est à noter qu'il peut très bien y avoir plus d'un chemin pour atteindre la destination avec un nombre minimal de déplacements. Le chemin « trouvé » va seulement dépendre de l'ordre dans lequel on va chercher dans notre algorithme récursif. Dans l'exemple précédent, si nous avions commencé à chercher par la gauche (au lieu de à droite), nous aurions trouvé un autre chemin :

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4

→

4	∞	0	1
3	2	1	2
4	3	∞	3
5	4	5-D	4