TP C# 10 - Follow me !

AVERTISSEMENT : tout code ne compilant pas attribuera immédiatement la note zéro au TP *entier*. Vérifiez sous Visual Studio que l'*error list* est bien vide avant de rendre.

1 Avant de commencer

1.1 Consignes de rendu

Vous devez faire un dossier du même nom que l'archive zip à rendre. Dans ce dossier doivent se trouver un fichier texte `AUTHORS`, ainsi que le dossier de votre projet. Vous n'avez pas besoin de rendre de fichiers .sln, .suo ou autres .cache.

Le fichier texte `AUTHORS` contiendra une étoile `*`, un espace, puis votre login (ex : login_x) suivi d'un retour à la ligne.

Par exemple, si vous vous appelez Xavier Login, votre archive pourrra, et **devra**, ressembler à la suivante :

```
rendu-login_x\AUTHORS
rendu-login_x\Astar.sln
rendu-login_x\Astar\*
```

2 Les listes chaînées

Nous allons aujourd'hui aborder une notion très importante qui vous suivra certainement durant une bonne partie de votre carrière. Tâchez de bien suivre ce TP et de bien en assimiler l'ensemble des notions.

2.1 Pourquoi des listes ?

Dans le monde informatique, et quel que soit le domaine – médical, jeux-vidéo, développement mobile – dans lequel vous évoluerez, les listes seront présentes.

Les ordinateurs traitent énormément de données dans des laps de temps restreints. C'est pourquoi il est important que ces données soit ordonnées, comprendre *rangées*. Les listes possèdent l'avantage d'être simples à manipuler ; de plus, elles peuvent subir des transformations souvent très utiles à la résolution de problèmes – notamment des opérations sur les tris, mais vous l'avez déjà vu en Caml.

2.2 Première approche

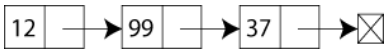
La liste est une représentation des données pour le programmeur. L'ordinateur ne connaît pas cette notion "de base", il faut lui apprendre.

La méthode la plus simple est de réunir toutes les informations de la liste dans un tableau, et de parcourir ce tableau selon les besoins. Cependant, cette implémentation possède différents défauts :

- La taille doit être fixée à l'avance. Imaginons que vous souhaitiez faire une liste de tous les étudiants étant venus à EPITA aujourd'hui ; pour être sûr que tout le monde tiendra dans le tableau, il vous faudra obligatoirement prévoir un tableau au moins égal au nombre d'élèves dans la promo. Et si seulement la moitié de la promo venait... Vous auriez alloué deux fois plus de place que besoin.
- Les opérations sont coûteuses. Imaginons maintenant que vous vous trouviez dans une liste triée, et que vous souhaitiez supprimer le premier élément en conservant la liste triée. La seule solution serait de supprimer l'élement, puis de tout décaler d'une case – pour éviter de laisser la première case vide. Pas très intéressant, quand ceci pourrait être fait en une seule *action*.

2.3 Chaînées ?

Pour contourner tous ces aspects, nous allons vous montrer une autre façon de représenter une liste dans un programme.



Considérez maintenant l'exemple ci-dessus. Il n'y a plus de tableau, mais seulement des éléments, qui en plus d'avoir une valeur, possèdent l'adresse de leur successeur.

Ainsi, si l'on souhaite parcourir cette liste pour rechercher un élément, il convient de retenir la tête – le premier élément – et de suivre ce protocole :

```
RECUPERER le premier élément

EST-CE l'élement recherché ?
OUI -> on s'arrête et on retoune YES
NON -> on cherche l'adresse de l'élement suivant, on s'y rend, et on ré-applique cette étape
jusqu'à se trouver sur un élément qui n'a pas de successeur -- pas d'adresse
```

Pour cela, nous manipulons directement des adresses mémoire par le biais de pointeurs. Cependant, le C# étant une "grosse" surcouche du C, il n'est pas possible d'utiliser ces pointeurs, tout ayant été simplifié et pré-implémenté.

C'est pourquoi nous allons vous demander de réaliser ce TP avec une liste normale, mais il est important que vous connaissiez l'existence des pointeurs et leur importance ; vous les manipulerez presque tout le temps dans quelques années.

3 L'algorithme A *

L'algorithme A* (prononcer A-Star) est l'un des plus célèbres algorithmes de recherche du plus court chemin. Il permet, sur un terrain défini, de trouver le chemin le moins *coûteux* pour se rendre d'un point initial vers un point destination.

3.1 Le coût d'un chemin

Ce qui fait la puissance et la réputation de cet algorithme, après sa simplicité d'implémentation, c'est la prise en compte du coût de déplacement.

Imaginons un personnage souhaitant se rendre de l'autre côté d'une forêt. Il existe deux solutions majeures :

- Traverser la forêt tout droit
- Contourner la forêt

Il est ici intéressant de savoir s'il est plus rapide de traverser la forêt ou de la contourner. Pour cela, nous allons fixer qu'un pas sur l'herbe a un coût de 1, et qu'un pas dans la forêt a un coût de 2 – il est en effet plus difficile et moins rapide de marcher en forêt.

Chaque déplacement possède donc un coût qui lui est propre. C'est notamment grâce à cette notion que l'algorithme A* est capable de vous donner une réponse très optimisée.

3.2 Open list et Close list

Cet algorithme va utiliser deux listes pour parvenir au résultat final.

Il se place au début sur la case initiale, et calcule la somme du coût de cette case et de la distance heuristique entre cette case et la case d'arrivée – la distance heuristique correspond à la distance la plus courte, sans prendre en compte les éventuels obstacles et coûts [1]. Après avoir attribué le résultat à la case, il ajout cette dernière à l'*open list*.

L'algorithme retire ensuite la première case de l'open list – ici la seule. S'il n'y en a plus à retirer, il n'existe aucun chemin. S'il s'agit du noeud d'arrivée, on reconstruit le chemin. Sinon, on applique la

1. http ://fr.wikipedia.org/wiki/Distance_de_Manhattan

même opération sur toutes les cases adjacentes à ce noeud, et ainsi de suite jusqu'à tomber sur l'un des cas décrits en début de paragraphe.

A chaque fois qu'une case est nouvellement visitée, elle est ajoutée à la *closed list*. Lors de la vérification d'une case, on vérifie tout d'abord si elle n'est pas déjà présente dans cette liste avec un coût inférieur ou égal ; si c'est le cas, on passe son tour et on continue avec les cases suivantes de l'open list.

Plus de détails et des illustrations directement sur la page Wikipédia de l'algorithme [2].

2. http ://fr.wikipedia.org/wiki/Algorithme_A*

4 FixMe //

Vous l'aurez devenir, nous allons vous demander de mettre en place cet algorithme. Cependant, il n'est pas simple de voir le rendu dans une console, et il est bien plus pratique d'avoir un personnage et des chemins pré-établis pour voir si le chemin emprunté est optimal.

Pour vous éviter de perdre du temps sur XNA, nous vous fournissons un template complet, parsemé de balises *//FIXME* à côté desquelles vous devrez insérer votre code.

Compilez et exécutez pour voir le personnage se déplacer, et constater ou non le bon fonctionnement de votre algorithme.

4.1 A vous de jouer

Nous vous indiquons à chaque fois l'endroit auquel vous trouverez la section de code à compléter, et ce qu'il est nécessaire d'y implémenter.

4.1.1 Distance de Manhattan

- Fichier : **Node.cs**
- Ligne : **31**

- Objectif : **implémenter la Distance de Manhattan telle qu'elle est décrite dans la partie cours du sujet**

4.1.2 Listes

- Fichier : **Pathfinding.cs**
- Ligne : **21**

- Objectif : **trouver et communiquer à l'algorithme les coordonnées du noeud de départ**
- Fichier : **Pathfinding.cs**
- Ligne : **27**

- Objectif : **ajouter le noeud de départ à l'open list**
- Fichier : **Pathfinding.cs**
- Ligne : **31**

- Objectif : **récupérer le premier élément qui contient le meilleur coût grâce à l'insertion dichotomique, et transférer cet élément de l'open list vers la closed list**
- Fichier : **Pathfinding.cs**
- Ligne : **40**

- Objectif : **ajouter le noeud en tête pour renverser la liste complète**
- Fichier : **Pathfinding.cs**
- Ligne : **45**

- Objectif : **récupérer la liste des cases adjacentes à la case en cours d'analyse**
- Fichier : **Pathfinding.cs**
- Ligne : **54** / **58** / **66**

- Objectif : **parcourir la liste des noeuds adjacents dans son intégralité afin de vérifier pour chacun qu'il n'est pas présent dans la closed list**

5 Conclusion

In ACDC you trust.