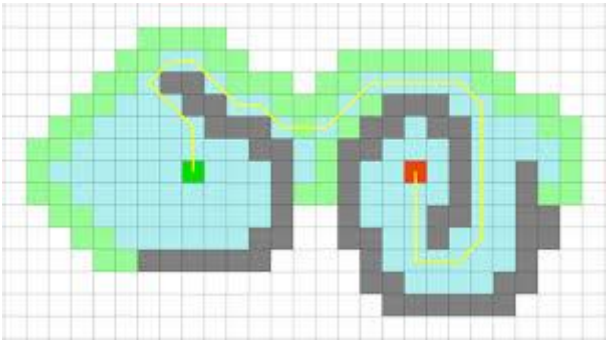
Partiel Second Semestre B1 Game : *Pathfinding* et Algorithme A*

- Durée : 4H
- Tous documents et web autorisés
- Compte gratuit GitHub requis

Introduction

Très utilisée dans le jeu vidéo, la recherche de chemin, couramment appelée *pathfinding*, est un problème de l'intelligence artificielle qui se rattache plus généralement au domaine de la planification et de la recherche de solution.

Il consiste à trouver comment se déplacer dans un environnement entre un point de départ et un point d'arrivée en prenant en compte différentes contraintes.



Deux algorithmes principaux sont utilisés pour la recherche du plus court chemin sur un graphe :

- **L'algorithme de Dijkstra**, qui permet de déterminer le chemin optimal. Il est, par exemple, utilisé pour le routage Internet ;
- **L'algorithme A***, qui est beaucoup plus rapide à condition d'avoir une bonne fonction heuristique, et dont l'optimalité n'est garantie que sous certaines conditions. En pratique, l'algorithme A* est un bon compromis entre coût de calcul et optimalité de la solution.

Vous pouvez faire l'essai visuel de ces algorithmes à cette adresse : http://qiao.github.io/PathFinding.js/visual/

Notes pour l'examen

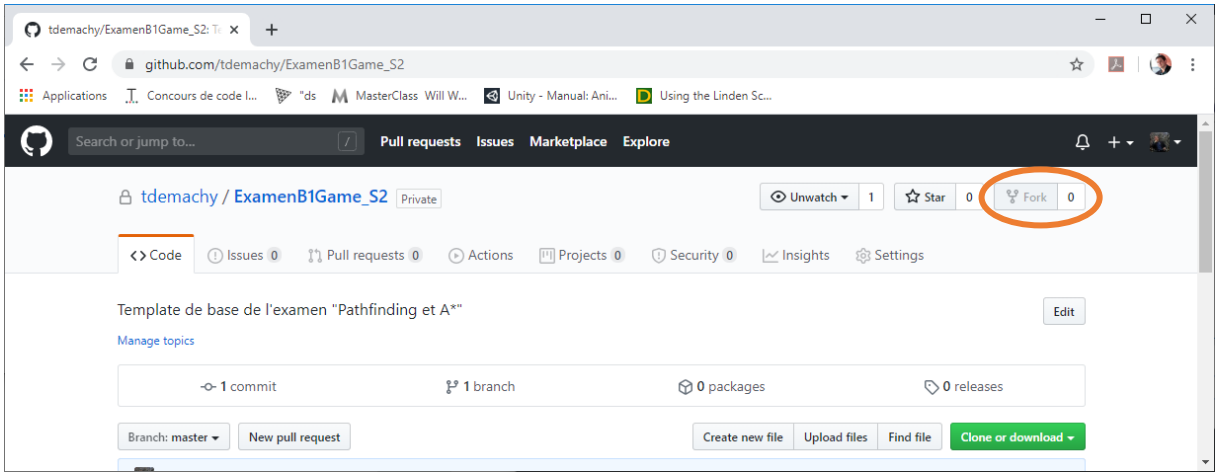
Cet exercice est un grand classique de l'informatique. Vous allez trouver sur le net de très nombreuses ressources. Vous avez accès a toutes. La seule chose que j'exige est de ne pas copier-coller du code et de rédiger vous-même tout votre code.

Récupération des données de base

La base du projet : énoncé, données test, début du projet sont disponibles à cette adresse :

https://github.com/tdemachy/ExamenB1Game_S2

Avant de cloner le projet il faudra vous en créer votre version. On appelle cela « Forker » un projet. Une fois connecté et à l'adresse donnée, cliquez sur le bouton entouré ci-dessous.



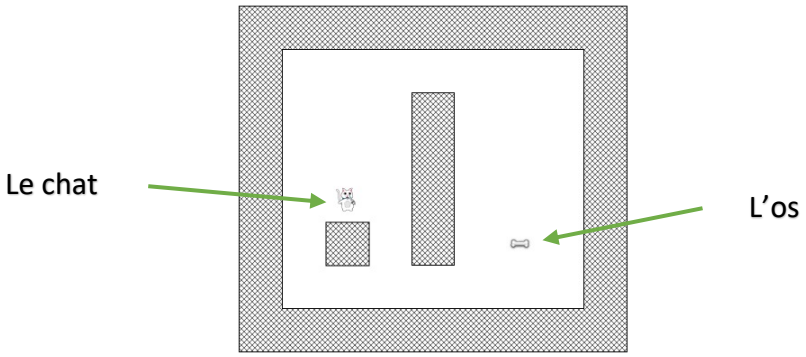
Cela créera votre propre version. C'est celle-ci que vous clonerez avant de travailler dessus.

Le rendu se fera sur GitHub, en poussant vos fichiers.

Si jamais vous avez un probleme - contactez moi !

Présentation de l'algorithme

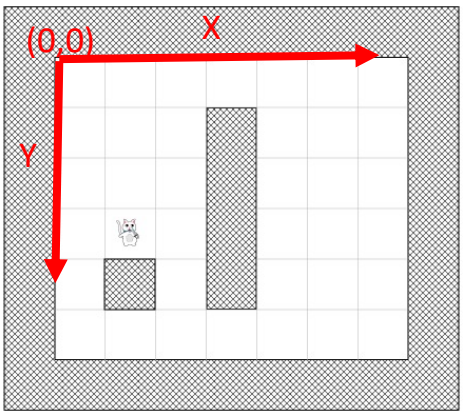
Supposons qu'un chat veuille retrouver son os. Vous allez me demander pourquoi un chat voudrait trouver un os, mais c'est un tout autre sujet.



Le principe est de calculer le chemin le plus simple de l'un à l'autre.

Pour simplifier le problème, la première approximation que nous allons faire et de superposer une grille à ce problème. X marquera l'axe horizontal des abscisses et Y marquera l'axe vertical des ordonnées.

L'origine (0,0) est le coin en haut à gauche.



Principe du A*

L'algorithme est itératif et recommence tant qu'il n'a pas atteint la case cible. Il s'appuie sur deux listes, une **liste ouverte** et une **liste close**. Dans la liste ouverte, on place les prochaines cases à évaluer. Dans la liste close, on y déplace les cases évaluées qui ne seront pas conservées.

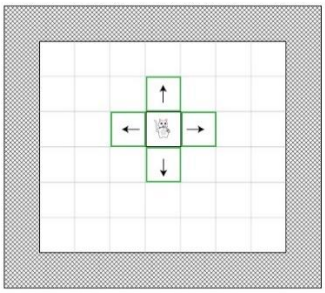
Le terme de liste utilisé ici est trompeur : elles peuvent être codées sous forme de tableau de taille suffisante ou de liste chainée.

Chaque case est codée dans une structure contenant un pointeur sur la case d'où l'on vit et les informations nécessaires au calcul (cf ci-dessous). On appelle cette structure un nœud (node en anglais)

Etape 1 : Trouver les cases accessibles

La case de départ est placée en liste close : il s'agit de ne pas revenir sur ses pas.

Pour chaque case dans la liste ouverte on regarde toutes les cases accessibles à partir de ces cases et on les ajoute en liste ouverte. Par exemple si notre chat peut se déplacer d'une seule case haut / bas ou gauche / droite , on ajoute les quatre cases à la liste « ouverte », et on leur donne pour parent la première case.

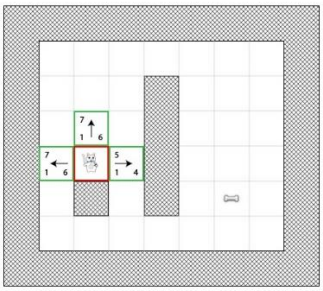


Etape 2 : Calcul F et G et H

Ce sont les données à calculer pour prendre de bons choix

- **G** est la distance depuis le point de départ : il est aussi égal au nombre d'itérations dans notre cas puisqu'on s'éloigne d'une case à chaque fois.
- **H** est la distance estimée au point d'arrivée. C'est le point le plus important de la méthode A*. H est une heuristique. Une heuristique est une méthode de calcul qui fournit rapidement une solution réaliste, pas nécessairement optimale ou exacte.
    - Dans notre cas, nous utiliserons pour H la distance de Manhattan, nommé après la ville construite en quadrillage. https://fr.wikipedia.org/wiki/Distance_de_Manhattan
    - Pour calculer la distance de Manhattan , il suffit d'ajouter le nombre de lignes au nombre de colonnes entre deux points, simple et rapide.
- **F** est la somme de G et de H,

Si on l'applique à notre problème, voilà le calcul de G et H (en bas de chaque case) et F (en haut d'une case)



Etape 3 : choix de la case suivante

On choisit dans la liste ouverte la case avec le F le plus faible. En l'occurrence c'est 5. Cette case est retirée de la liste ouverte pour passer en liste close et on recommence l'étape 1, jusqu'à ce qu'on mette la case d'arrivée dans la liste close.

Etape 4 : définir le chemin

En partant de la case destination puis en remontant, case par case, de parent en parent, jusqu'à la source. Ces cases définissent le chemin. On cherche à les compter.

Organisation du fichier de test

Deux fichiers vous sont livrés :

- Le fichier de grille
- Le fichier de tests

Grille

Le premier est grid.txt : c'est la grille. Elle fait 10 x 10. Dans le fichier, il y a donc dix lignes de dix caractères. Chaque caractère est un 1 si la case est accessible et un 0 si la case est inaccessible.

Test

Le fichier data.txt est le fichier de test son organisation est simple :

- Sur la première ligne il y a le nombre de tests a effectués
- Sur la seconde, 5 valeurs : x, y du départ x, y de la fin et le nombre de cases du parcours A*. C'est cette dernière valeur que vous utilisez pour valider vos résultats.

Travail demandé

Lecture des données (est. 1h)

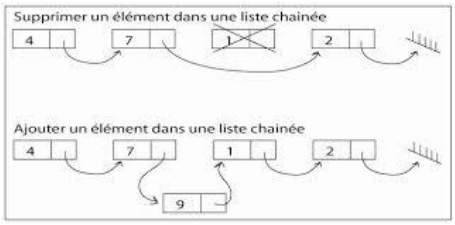
- Définir une grille : votre grille est un tableau à deux dimensions (hauteur * largeur) d'entier. 1 pour une case accessible et 0 pour une case occupée.
- Lire la grille à partir du fichier de test (cf « organisation du fichier de test »)
- Lire la position de départ et de destination dans le fichier de test
- Lire la solution (distance parcourue en nombre de cases entre départ et destination)

Construction des listes (est. 30 min)

- Définir la structure Node (nœud). Proposition pour cette structure :
  - **parent** : Un pointeur sur une structure node
  - **next** : un pointeur sur une structure node, si utilisation d'une liste chainée, le chainage passera par cette valeur
  - **f, g, h** : entiers contenant la valeur de chaque nombre
- Définir la structure List qui contient
- Un pointeur sur le premier node
- Déclarer la liste ouverte et la liste close

Ajouter des fonctions push et remove (est. 30mn)

- Push ajoute un nœud a la fin d'une liste : on parcours la liste jusqu'à ce que next soit nul, puis on assigne un pointeur sur le nœud à rajouter à next.
- Remove retire un nœud d'une liste : on parcourt la liste jusqu'au nœud a supprimer en gardant en mémoire a chaque fois le précèdent, puis on remplace le next du précédent par celui de l'élément à supprimer. Et hop



Ecrire l'algorithme du A*

On ne peut aller que sur les cases haut, bas, gauche, droite, les diagonales sont interdites

- Suivre l'algorithme étape par étape
  - Etape 1 – est. 30 min
  - Etape 2 – est. 15 min
  - Etape 3 et boucle – est. 45 min
  - Etape 4 – estimation. 30 min
- Afficher le chemin (nœud par nœud, chaque coordonnées) et le résultat : la taille du parcours, le nombre de cases
- Comparer le résultat à celui dans le fichier de test.