Activité intégrative 2024

Algorithme de détection d'un chemin entre deux bords

Ce document présente un algorithme pour détecter un chemin liant deux des trois bords. Cet algorithme intervient dans la détection d'une fin de partie.

Important

Dans la suite, nous décrivons ce que doit faire l'algorithme. Certains détails pourraient manquer. Il est de votre ressort d'essayer de comprendre l'algorithme pour qu'il fonctionne correctement (à vos crayons !).

N'oubliez pas de *continuer à respecter les principes de la programmation orientée-objet* lors de l'implémentation de votre algorithme !

Données de départ

L'algorithme reçoit les coordonnées C_j des tuiles qu'un joueur occupe. Pour chaque coordonnée C_{ij}, on peut calculer ses coordonnées voisines, c'est-à-dire celles dont la distance avec C_{ij} vaut 1.

Données de sortie

L'algorithme retourne la longueur du chemin trouvé ou un nombre négatif dans l'absence d'un tel chemin.

Idée de l'algorithme

L'idée fondamentale de l'algorithme est de visiter les coordonnées occupées par le joueur en commençant par les coordonnées situées au bord du plateau. En incorporant les autres coordonnées dans un ordre *précis*, on peut détecter si deux bords sont connectés tout en calculant la longueur du chemin qui les relie. Forcément, l'algorithme doit mémoriser quelles coordonnées ont déjà été visitées afin d'éviter d'en tenir compte plusieurs fois. À chaque coordonnée rencontrée, on associe deux informations :

- une indication du bord par lequel on l'a découverte ;
- la distance de cette tuile à ce bord ;

Étapes de l'algorithme

L'algorithme a potentiellement besoin de plusieurs collections intermédiaires. Nous ne vous dirons pas exactement lesquelles. À vous de décider de quelles interfaces et implémentations concrètes vous avez besoin :

- Les coordonnées de départ, reçues en argument, appelée dans la suite « `coordonnées` » ;
- Une collection pour retenir les coordonnées qui ont déjà été vues, appelée dans la suite « `dejàVues` » ;

– Une collection qui regroupe les coordonnées dont il faut aller visiter les voisins, appelée dans la suite « `voisinsAVisiter` ».

Important

Essayez de comprendre l'algorithme avant de vous lancer dans son implémentation. Examinez chacune des collections dont vous avez besoin : Doivent-elles être triées ? Doivent-elles implémenter une politique d'accès particulière (LIFO, FIFO…) ? Devez-vous associer des informations ?

Initialisation

Sur base de `coordonnées`, calculer les coordonnées situées sur un des bords. La distance de ces tuiles à leur bord est donc de 0. Retenir qu'on les a `dejàVues` et les ajouter également dans `voisinsAVisiter`.

Boucle

Tant qu'on n'a pas trouvé un chemin connectant deux bords et qu'il reste des coordonnées dans `voisinsAVisiter` :

1. Retirer une coordonnée `c` de `voisinsAVisiter` dont la distance à un bord est la plus petite ;
2. Examiner les coordonnées voisines de `c` (et de la même couleur) :
    a. Si l'un des voisins a déjà été vu et provient d'un autre bord que `c`, cela veut dire que vous avez trouvé votre chemin reliant les bords (`c` se trouve d'ailleurs, à une case près, au milieu de ce chemin). Puisque vous connaissez la distance au bord de `c` et de son voisin, calculer la longueur du chemin ne devrait pas vous poser de problème.
    b. Sinon, pour chacun des voisins qu'on vient de découvrir, il faut se rappeler qu'on l'a déjà vu. Il a été découvert par le même bord que `c` (mais il faut traverser `c` pour l'atteindre). Il faut aussi faire en sorte de se rappeler qu'il a des `voisinsAVisiter` (et donc il faut l'intégrer intelligemment à cette collection).

Fin

Ne pas oublier de retourner une valeur négative si aucun chemin entre des bords n'a été trouvé.

Important

Cet algorithme semble intéressant mais rien ne dit qu'il se termine. À vous d'en déterminer sa *fonction de terminaison* (si vous ne savez pas ce que c'est, jetez un coup d'œil dans votre cours d'algorithmique)

Exemple

La Figure 1 présente un plateau de jeu dans lequel on s'intéresse aux cases jouées par le joueur rouge. Nous avons appelé le bord en haut à gauche « S » ; celui en haut à droite « Q » et celui du bas « R ». La Figure 2 présente le résultat de l'algorithme. La case en jaune « Q4 »

est la dernière qui sera envisagée par l'algorithme. En effet, elle possède un voisin qui provient d'un bord différent (R4). Il reste deux cases qui ne seront pas vues lors des itérations : ce n'est pas grave. L'algorithme renverra la valeur « 10 ». Note : l'algorithme pourrait tout aussi bien regarder les voisins de R4 et trouver Q4 parmi eux. Il renverra de toute façon le même résultat.

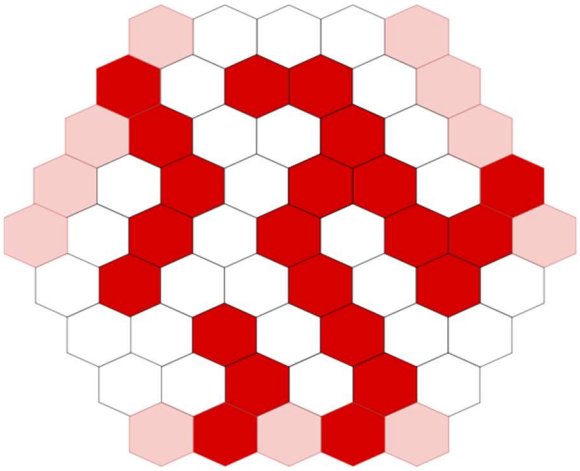

FIGURE 1 : AVANT L'EXÉCUTION DE L'ALGORITHME

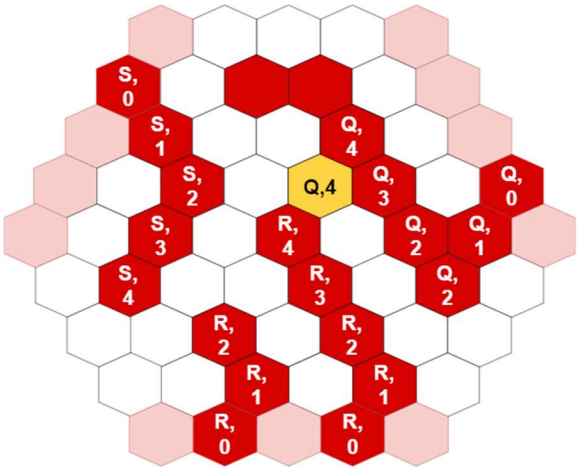

FIGURE 2 : APRÈS L'EXÉCUTION DE L'ALGORITHME

2[d] exemple

La Figure 3 présente un plateau de jeu dans lequel on s'intéresse aux cases jouées par le joueur bleu. Dans la Figure 4, nous avons appelés R, S et Q les côtés ou la composante vaut $-rayon$. Notez qu'appeler ainsi vos côtés n'est pas obligatoire mais facilite nos explications.

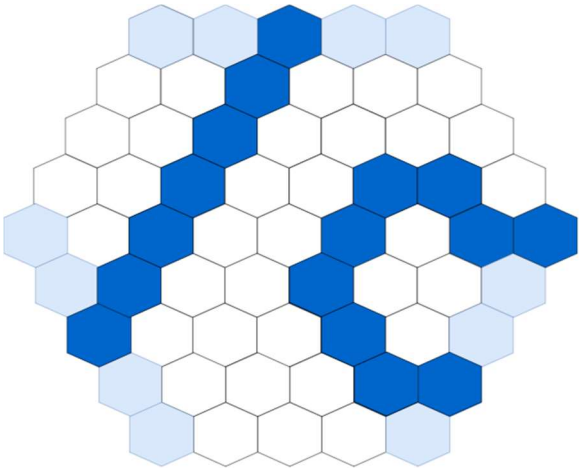

FIGURE 3 : AVANT L'EXÉCUTION DE L'ALGORITHME

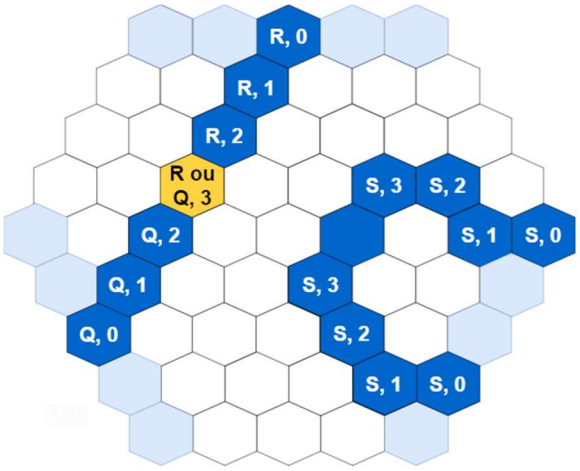

FIGURE 4 : APRÈS L'EXÉCUTION DE L'ALGORITHME

Suivant la manière dont votre algorithme a inséré et retiré les positions dans vos collections, la case jaune sera notée R3 ou Q3. Cela n'a pas d'incidence sur la valeur renvoyée par l'algorithme : 7.