Bruno ARIGANELLO
Titouan CORNILLEAU

Projet de Théorie des jeux
- Le jeu de Hex -

Université de Rouen
MASTER 1 GIL-ITA

Sommaire

I - Manuel d'utilisation

Pour lancer l'application, il vous faut exécuter le fichier .jar fourni dans l'archive transmise. Le jeu démarre dès l'apparition de la fenêtre. Vous pouvez utiliser les boutons sur la barre de menu pour afficher les règles du jeu, l'histoire de sa création, ainsi que pour commencer une nouvelle partie.

Le jeu de Hex est un jeu de société combinatoire abstrait pour deux joueurs. Il se joue sur un tablier en forme de losange dont les cases sont hexagonales.

Le joueur blanc commence. Les joueurs jouent chacun leur tour. À chaque tour, le joueur place un pion de sa couleur sur une case libre du plateau. Le premier joueur qui réussit à relier ses deux bords par un chemin de pions continus de sa couleur a gagné. Il ne peut y avoir qu'un pion par case. Les pions posés le sont définitivement, ils ne peuvent être ni retirés,ni déplacés.

II - Structure de donnée

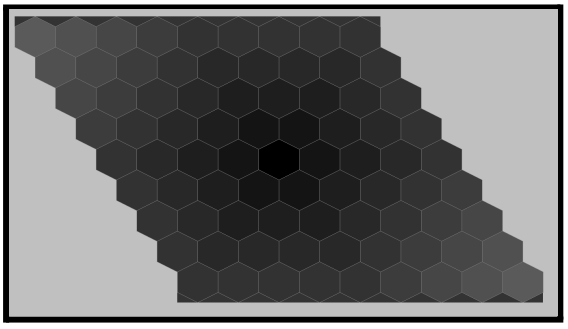
Pour représenter les nœuds et les graphes, nous avons défini des objets java Nodes. Ceux-ci vont contenir les différentes informations qui seront utilisées dans les algorithmes de traitement. Ces attributs sont variés, et vont de l'approximation f(x) à la couleur en passant par la liste des voisins.

III - L'heuristique

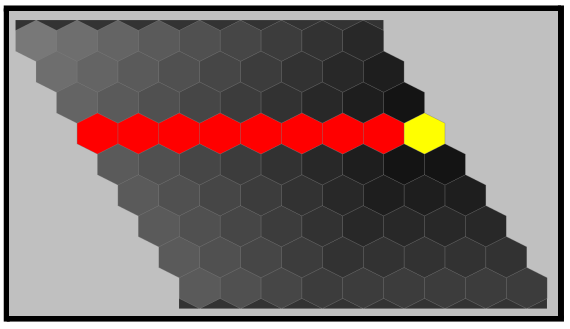
Dans le cadre du problème du jeu de Hex, nous avons écrit une méthode pour définir une heuristique. Cette méthode est fonctionnelle mais nous nous sommes rendu compte qu'elle n'était pas pertinente avec notre problématique. Le but de l'heuristique est de paramétrer l'algorithme pour prendre en compte des détails propres à un problème.

Par exemple, si nous devons modéliser une carte montrant des villes reliées par des routes, une heuristique peut être utilisée pour préciser que les distances à vol d'oiseau entre chaque ville n'est pas fondamentalement plus rapide à traverser que les distances sur routes. Le problème est que, dans le jeu du Hex, chaque nœud est relié à six autres voisins, eux-mêmes reliés à six autres voisins (sauf ceux en bord de plateau). De plus, chaque arc (c'est-à-dire chaque distance entre deux nœuds voisins) a pour valeur 1.

Cela rend l'usage d'une heuristique inutile pour résoudre notre problème du plus court chemin. C'est pourquoi nous ne nous sommes pas servis de cette fonction, bien que nous l'ayons laissé dans notre code. Chaque nœud possède donc une heuristique statique de zéro.



À gauche, une modélisation du calcule de l'heuristique depuis le pion central. Plus un pion est clair, plus son heuristique est grande.



À droite, une modélisation de notre calcul du plus court chemin depuis le pion jaune. L'heuristique fut calculée également.

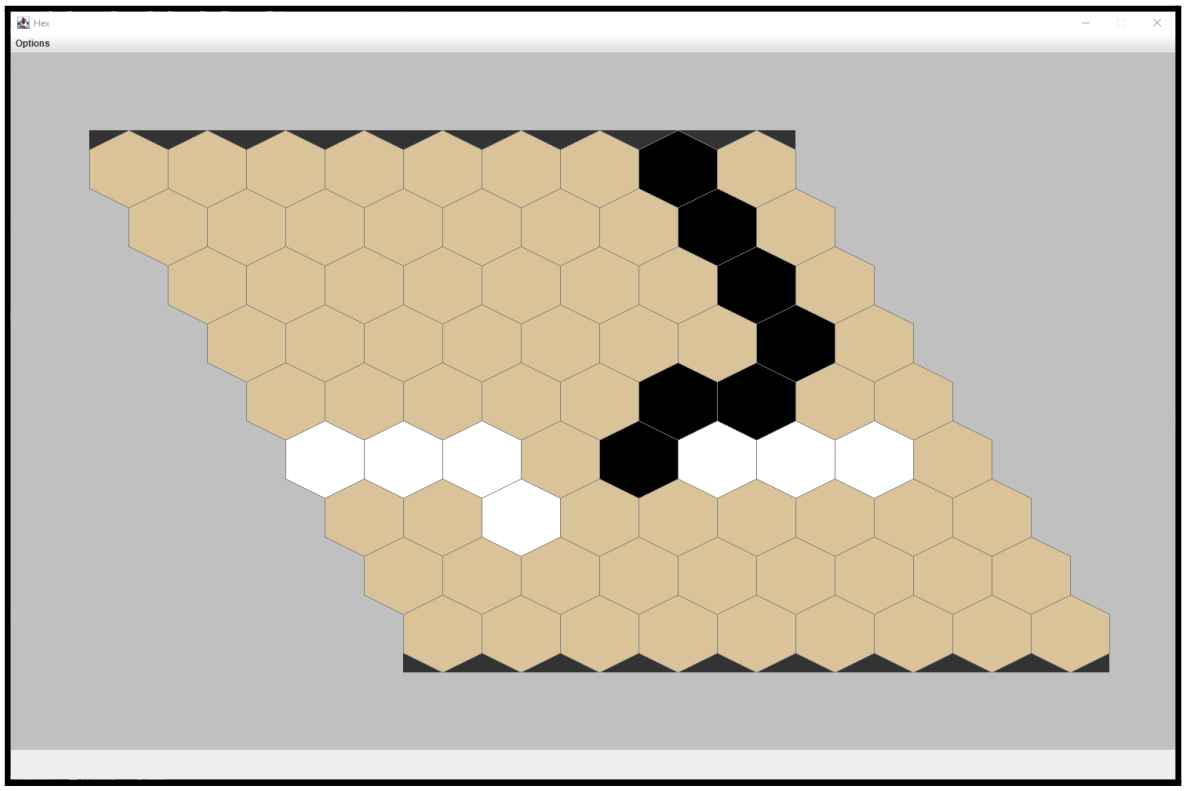
IV - Algorithme A*

Pour examiner les différents chemin possible, et choisir le meilleur, nous avons implémenté l'algorithme **A***, une variante de l'algorithme de Dijkstra. A∗ calcule une approximation d'un plus court chemin de s vers un sommet destination d, en utilisant une heuristique h connue à l'avance.

Pour tout sommet x, h(x) est une estimation d'un plus court chemin entre x et d. On associe, en outre, à tout sommet x, une approximation f(x) qui tient compte de h(x) et on examine, en priorité, un sommet ouvert d'approximation minimale.

V - Condition de victoire

Pour vérifier si la victoire a été atteinte par le joueur ou l'IA, nous vérifions à chaque action si un chemin existe entre les deux extrémités du plateau. Pour ce faire, nous avons utilisé une version modifié de l'algorithme A*. Dans cette version, nous n'observons que les nœuds de la même couleur que le nœud source, et nous ne cherchons pas à obtenir le chemin le plus court, simplement le premier que l'on obtient.



L'IA est en noir, le joueur en blanc