**Algorithmes utilisés**

* 1. Arbre de Monte Carlo / algorithme min-max / Algorithme
  2. Arbre de Monte Carlo / UCTS / RAVE / AMAF

**Idées pour l'IA**

Position qui assure un lien (bloquer automatiquement lorsque l'utilisateur essaye de casser ce lien ?) :

* 1. Losange (entre deux pions)
  2. Ziggurate (avec le bord)

Position ou un seul coup est possible :

* 1. En défense (s’il y a défaite assuré si un coup n'est pas joué)

Jouer automatiquement si victoire à 100% ou s'il y a un seul coup possible pour éviter la défaite

Highlight Certaines connaissances peuvent ne pas être bénéfique pour l'IA (éviter d'implémenter des stratégies manuellement) car elles ne prennent pas en compte tous les pièges qui peuvent être posés

**Gestion des données**

**Cases mortes**

Existence des cases mortes (pas besoin d'être évalué) => tableau de booléen

**Pions**

|  |  |  |  |
| --- | --- | --- | --- |
| Idées | 1 seule tableau pour tous les pions avec 3 valeurs possible par cases :   * 1. Vide = 0   2. Blanc = 1   3. Noir = 2 | 2 tableaux (1 par couleur) :   * 1. On rentre que les pions qui ont déjà été placé | 2 Listes / Piles (1 par couleur) |
| Avantages | Plus adapté à la méthode du circuit électrique ? | Adapté à la méthode du circuit électrique | Permet de tester récursivement si un joueur gagne |
| Inconvénients | Plus long pour différencier les pions blancs/noirs |  | Stack Overflow possible |

**Connexions**

Enregistrer les connexions virtuelles / semi-virtuelles / réelles dans 3 tableaux => évite de les recalculer à chaque fois

Highlight Ne pas oublier de les supprimer si elles sont détruites

**Fonction d'évaluation**

**Méthode du circuit électrique :**

Contact Semi-connexions / connexions (voir PDF fonction d'évaluation)

Definition Connexion virtuelle : noir peut relier ses deux pièces même si blanc commence

Definition Semi-connexion virtuelle : noir peut relier ses deux pièces s'il commence

Definition La profondeur d'une connexion est la profondeur de l'arbre requis pour établir la connexion

Position initiale = Semi connexion (car stratégie gagnante)

**H-search pour trouver les connections virtuelles/semi-virtuelles**

On considère et deux connexions virtuelles avec et deux cases noirs, et deux ensembles de cases vide qui relie deux cases ( et pour et et pour ) On prend et et

Definition AND :

* 1. Definition Si est noir, alors et sont reliées par une connections virtuelle
  2. Definition Si est vide, alors et sont reliées par une connexion semi-virtuelle

Definition OR : Si et sont reliés par plusieurs connexions semi-virtuelles qui ne se chevauchent pas, alors et forment une connexion virtuelle

On utilise les propriétés AND et OR pour trouver une première génération de liens, on fait ça récursivement jusqu’à ce qu'il n'y ai plus de liens ou un lien gagnant

**Fonction d'évaluation**

2 circuits électrique dans le jeu (un pour les noirs et un pour les blancs)

Chaque case est associée à une résistance (exemple pour les noirs) :

* 1. 1 si la case est vide
  2. 0 si la case contient un pion noir
  3. si elle contient un pion blanc

Connexion virtuel traité comme cases voisines

Connexion entre 2 vrai voisin = plus fort que des connexions virtuelles => résistance moins forte

Plus la profondeur de la connexion est grande, plus la résistance est élevée

* Fonction : avec la résistance des noirs et la résistance des blancs.

Programme :

Fonction victoire :

Le principe est de regarder tous les chemins possibles à partir d’un bord du plateau. On regarde récursivement à partir des pions au bord si le pion suivant gagne. Pour cela, on stocke les pions voisins dans une pile, puis on rappelle la fonction avec chaque pion de la pile. Si un pion arrive à la dernière ligne / dernière colonne, le joueur a gagné. Si tous les pions de la pile ont été testé, on revient en arrière. Si on a testé tous les pions au bord, le joueur n’a pas gagné. Si un pion n’est pas relié avec le bord, il ne sera pas testé donc la fonction ne pourra pas renvoyée une victoire même s’il est sur le bord opposé.

IA 1.0 :

On simule aléatoirement jusqu’à la fin de la partie des milliers de fois. On enregistre dans un tableau le nombre de victoire par coup. Celui qui en a le plus sera joué. Le niveau est très faible (à peine plus élevé que celui d’un ordinateur qui joue aléatoirement) et l’IA met beaucoup de temps à jouer. Elle met entre 6 et 8 secondes en début de partie pour 100 000 tests sur un plateau 11 \* 11. C’est assez long car elle n’explore qu’une toute petite partie des parties possibles (122!). Ce qui fait que son niveau est très faible.

IA 1.5 :

On simule toujours aléatoirement jusqu’à la fin de la partie des milliers de fois mais on le fait différemment. On liste tous les coups possibles dans un tableau, on mélange ce tableau et on distribue entre les deux joueurs de manière à ce que le nombre de coups de chaque joueur soit respecté. L’IA est plus rapide et tests de manière équivalente toutes les cases. Pour 1000 tests par cases (122 000 en début de partie soit plus que l’IA 1.0), un coup met environ 5 secondes pour se faire. Le niveau est cependant toujours faible car l’évolution est beaucoup trop petite.

IA 4.0 (03/12/2023) :

On revient sur l’algorithme MCTS. On teste équitablement toutes les cases possibles. Pour 1000 tests par cases (un total de 121 000 tests par coup), l’IA met environ 45 secondes pour réfléchir. Son niveau est bien plus élevé que les versions précédentes mais reste faible comparé à un humain qui connaît un peu le jeu. On observe qu’elle essaye de créer un chemin dans la bonne direction lorsqu’elle commence. Lorsqu’elle est en situation de désavantage, elle essaye de bloquer l’adversaire comme un humain débutant ferait. Elle ne choisit pas la meilleur façon de bloquer mais elle en choisit une, ce qui est mieux que les versions précédentes où elle continuait de créer son chemin même si l’adversaire avait un coup gagnant.