État d'art d'algorithmes d'IA

1. MinMax

Principe :

Visite de l'arbre de jeu pour faire remonter à la racine une valeur qui est calculée récursivement ainsi :
Soit h heuristique,
- Si p terminal : MinMax(p) = h(p)
- Si p position du joueur : *soit {o1,…,on} fils de p,*
MinMax(p) = max(MinMax(o1), ..., MinMax(on))
- Si p position de l'opposant : *soit {j1 ,…,jm} fils de p,*
MinMax(p) = min(MinMax(j1), ..., MinMax(jm))

Avantage(s) :

- Facilité d'implémentation
- MinMax(p) = Meilleure valeur possible à partir de la position p si l'opposant joue de façon optimale

Inconvénient(s) :

- Nécessite une fonction d'évaluation (fonction rapide qui associe une évaluation de sa valeur par rapport à sa position) → Pas possible pour le Blokus (trop de cas différents)
- Coût exponentiel

---

2. Alpha-Beta

Principe :

Optimisation de l'algorithme MinMax :
Stoppe l'itération des sous-arbres dont la valeur est jugée inintéressante en vue du calcul de la valeur MinMax du jeu.

A chaque nœud : Etudie valeur du nœud, valeur alpha, valeur beta où :

■ alpha :
→ Initialisée à -Infini
→ Sur les feuilles : valeur de la Feuille
→ Sur les noeuds : + **grande** valeur obtenue sur les fils visités précédemment & = valeur alpha de son prédécesseur sur les nœuds associés à l'opposant

■ beta :
→ Initialisée à +Infini
→ Sur les feuilles : valeur de la Feuille
→ Sur les noeuds : + **petite** valeur obtenue sur les fils visités précédemment & = valeur beta de son prédécesseur sur les nœuds associés au joueur

Avantage(s) :

- Optimisation du MinMax → division du temps d'exécution par 2

Inconvénient(s) :

- Nécessite une fonction d'évaluation (fonction rapide qui associe une évaluation de sa valeur par rapport à sa position) → Pas possible pour le Blokus (trop de cas différents)

---

3. Algorithme de Monte-Carlo

Principe :

- Simulation d'un grand nombre d'actions disponibles en jouant de manière aléatoire à partir d'un état du jeu, et ce jusqu'à une situation terminale.
- Evalue le score moyen de chaque action selon un score de victoire/défaite
- Choisit l'action dont le score moyen est le plus élevé → Meilleure décision

Avantage(s) :

- Correspond au type de jeu qu'est le Blokus
- Résultats numériquement bons (c'est-à-dire grande probabilité de gagner contre un algorithme séquentiel)

- Temps d’exécution déterministe
- Adapté à des environnements dynamiques

Inconvénient(s) :

- Moins compétent contre des êtres humains (même s’il l’est toujours)
- Très dépendant des paramètres d’entrée et de la modélisation (pas d’erreur admise)

Version performante au Blokus :
Monte Carlo Tree Search (MCTS)

---

4. Wave Algorithm / Algorithme de la vague

Principe :

A partir d’une situation de départ, explore tous les voisins et leur attribue un score croissant selon leur distance à la source.
On remonte les valeurs décroissantes depuis la cible jusque la source pour avoir le chemin optimal

Avantage(s) :

- Très simple à implémenter
- Garantit le choix optimal

Inconvénient(s) :

- Adapté aux recherches de chemin, donc pas au jeu du Blokus qui est plus complexe
- Temps d’exécution long
- Coût en mémoire important

---

5. Dijkstra

Principe :

- Attribue une distance initiale infinie à chaque nœud et une distance de 0 à la source
- File de priorité :
Sélectionne le nœud voisin non visité de + petite distance
Mise à jour des distances calculées en passant par ce nœud
Marquage des nœuds visités au fur et à mesure

Avantage(s) :

- Garantit le chemin le plus court et le plus optimisé
- Meilleur que le Wave Algorithm

Inconvénient(s) :

- Non adapté au jeu du Blokus
- Non adapté aux environnements dynamiques (il faut tout recommencer à chaque fois)

---

6. A*

Principe :

Optimisation de l'algorithme de Dijkstra :
Valeur g = coût exact depuis la source jusqu'à la position actuelle
Heuristique h calculée à chaque position. Si celle-ci ne se rapproche pas du résultat escompté, on abandonne la solution pour retourner en arrière, afin de tester d'autres chemins possibles.

Avantage(s) :

- Garantit le chemin le plus court

- Plus performant que l’algorithme de Dijsktra (diminution du temps d’exécution)

Inconvénient(s) :

- Non adapté au jeu du Blokus
- Non adapté aux environnements dynamiques (il faut tout recommencer à chaque fois)
- Coût en mémoire élevé

---

7. Q-Learning

Principe :

Algorithme de Reinforcement Learning
Décisions prises dans l’objectif de maximiser la valeur d’une fonction Q (où Q est une fonction en lien avec l’algorithme du jeu)

Avantage(s) :

- Peut devenir meilleur grâce à l’exploration

Inconvénient(s) :

- Nécessite beaucoup d’exécution pour commencer à être performant
- Peu adapté au Blokus, qui exige une exploration massive d’états avant d’avoir de bonnes stratégies → entraînement trop long

---

8. Deep Reinforcement Learning (DRL)

Principe :

Combinaison du Deep Learning et du Reinforcement Learning

Réseaux de neurones pour approximer des fonctions de valeur

Avantage(s) :

- Adapté à de très grands espaces d’état
- Apprentissage à partir de l’exploration continue

Inconvénient(s) :

- Difficile à ajuster selon les applications
- Demande beaucoup de données et de puissance
- Adapté au Blokus uniquement avec des ressources massives pour parcourir tous les états