2024

ESILV

RAPPORT BATTLE IA

Cyprien Duceux
Kaveen Chanemougam
Abdoulaye Diop

SOMMAIRE

CONTEXTE ET OBJECTIFS

Au cours de notre projet, nous nous sommes attelés à la conception et au développement d'une Intelligence Artificielle pour jouer au jeu de Puissance 4. Notre objectif était de créer une IA capable de concourir efficacement contre un utilisateur dans un premier temps, puis contre les IA créées par les membres de notre groupe de TD. Nous avons utilisé des stratégies avancées et des algorithmes de décision. Ce rapport détaille le processus que nous avons suivi, les défis rencontrés et les solutions que nous avons mises en place pour développer cette IA.

Dans la première phase du projet, nous avons analysé différentes approches pour implémenter l'IA, en nous concentrant spécifiquement sur l'algorithme Minimax avec élagage Alpha-Beta (adapté pour les prises des décision dans les algorithmes à deux joueurs). En séparant l'équipe en trois sous effectif nous avons pu créer indépendamment trois modèles avec des logiques proches mais des fonctionnement très différents. A chaque étape d'optimisation nous avons comparé les retour de test de chaque modèle afin optimiser nos choix.

CONCEPTION DE L'ALGORITHME

- **Choix des bibliothèques**

Dans le développement de notre jeu de Puissance 4, nous avons utilisé plusieurs bibliothèques Python pour leurs fonctionnalités spécifiques qui répondent aux besoins du projet.

Numpy

Nous avons choisi numpy pour sa capacité à manipuler des tableaux multidimensionnels. Cette bibliothèque est essentielle pour la création et la gestion du plateau de jeu, qui est représenté sous forme d'un tableau 2D. Nous avons utilisé **numpy** notamment pour initialiser le plateau de jeu avec **np.zeros((LIGNES, COLONNES))** créant ainsi un tableau de 6 lignes et 12 colonnes.

Ensuite, afin d'jouter un élément d'aléatoire dans le jeu, notamment pour décider quel joueur commence, nous avons utilisé la bibliothèque **random**. Cette utilisation est illustrée par l'emploi (dans une des version du code) de **random.randint(0, 1)** pour choisir aléatoirement entre le joueur humain et l'IA comme premier joueur. Cela contribue à rendre chaque partie unique et imprévisible.

Enfin, nous avons fait appel à la bibliothèque **math**, particulièrement pour son utilité dans l'implémentation de l'algorithme Minimax avec élagage Alpha-Beta. Par exemple, l'utilisation de **math.inf** nous a permis de définir des valeurs initiales pour les variables alpha et beta dans cet algorithme, facilitant ainsi l'évaluation des meilleurs coups possibles pour l'IA.

CONCEPTION DE L'ALGORITHME

- **Les fonctions de jeu**

Nous avons implémenté la fonction ***placer_pion*** pour modifier directement l'état du plateau, rendant les changements persistants et visibles pour tout le programme. La fonction ***prochaine_ligne_libre*** utilise une approche itérative pour identifier la première ligne libre d'une colonne, retournant un indice ou ***None***. Cela influence directement l'utilisation d'autres fonctions comme ***placer_pion***.

La fonction ***afficher_plateau***, grâce à ***np.flip***, permet une visualisation intuitive du plateau en reflétant la gravité. ***coup_gagnant*** vérifie les alignements gagnants de pions et retourne un booléen, simplifiant la détermination de l'état de la partie.

Enfin, ***colonnes_valides*** identifie les colonnes disponibles pour jouer, influençant les décisions stratégiques de l'IA, notamment dans l'algorithme ***Minimax***.

```python
def placer_pion(plateau, ligne, colonne, pion):
    """Place un pion dans le tableau."""
    plateau[ligne][colonne] = pion

def prochaine_ligne_libre(plateau, colonne):
    """Trouve la prochaine ligne libre dans une colonne."""
    for r in range(LIGNES):
        if plateau[r][colonne] == 0:
            return r
    return None

def afficher_plateau(plateau):
    """Affiche le plateau de jeu."""
    for ligne in range(5, -1, -1):
        for colonne in range(12):
            print(int(plateau[ligne][colonne]), end=' ')
        print()
    print()
```

CONCEPTION DE L'ALGORITHME

- **La fonction Minimax**

Lors de la conception de la fonction ***minimax***, nous avons tout d'abord déterminé les paramètres essentiels du jeu, tels que le plateau, la profondeur de recherche, ainsi que les indicateurs alpha et beta pour l'élagage. Nous avons rapidement intégré une vérification des états terminaux du jeu et défini des valeurs spécifiques pour les cas de victoire, de défaite ou d'absence de mouvements.

Notre attention s'est particulièrement portée sur **l'élagage Alpha-Beta**. En intégrant cette méthode, nous avons significativement amélioré l'efficacité de l'algorithme en évitant d'explorer des branches moins prometteuses de l'arbre de décision. Cette optimisation était cruciale pour accroître la rapidité et la performance de notre IA.

Pour les cas de coups gagnants, nous avons attribué des valeurs extrêmement élevées. Nous avons remarqué que l'IA privilégiait parfois des **coups défensifs** au lieu de saisir des opportunités de victoire plus stratégiques. En ajustant les valeurs retournées pour les **coups gagnants** au sein de la fonction ***coup_gagnant*** (qui comporte nos quatre heuristiques; victoire **en ligne**, **en colonne**, **en diagonale** et en **contre diagonale**) couplée à la fonction ***evaluer_position***, nous nous sommes assurés que l'IA priorise toujours un coup gagnant lorsqu'il est disponible, ce qui a renforcé son approche stratégique du jeu.

En résumé, notre implémentation de ***minimax***, avec l'élagage Alpha-Beta et une pondération stratégique des valeurs de retour pour les coups gagnants, a permis à l'IA de prendre des décisions de jeu plus efficaces, en privilégiant les coups gagnants et en améliorant son gameplay global.

STRATÉGIE ET OPTIMISATION

Dans notre fonction minimax, nous avons initié le processus avec: **colonnes_possibles = colonnes_valides(plateau)** et; **est_terminal = est_noeud_terminal(plateau)**.
Ce choix a été stratégique pour déterminer rapidement les actions possibles et savoir si le jeu était dans un état terminal, évitant ainsi des calculs inutiles.

```
if coup_gagnant(plateau, joueur_ia):
    return (None, 100000000000000)
elif coup_gagnant(plateau, joueur_humain):
    return (None, -100000000000000)
```

L'attribution de grandes valeurs pour les coups gagnants était essentielle. Dans nos premiers tests, nous avions observé que l'IA, sans ces **valeurs extrêmes**, manquait parfois des opportunités de gagner en se **concentrant trop sur la défense**. Ces valeurs garantissent que l'IA **saisira toujours une opportunité de gagner** si elle se présente.

```
for col in colonnes_possibles:
    ligne = prochaine_ligne_libre(plateau, col)
    copie_plateau = plateau.copy()
    placer_pion(copie_plateau, ligne, col, joueur_ia)
```

Nous avons choisi de travailler avec une copie du plateau à chaque itération pour ne pas altérer l'état original du jeu. Cette approche nous a permis de tester différents scénarios sans affecter le déroulement réel de la partie.

STRATÉGIE ET OPTIMISATION

```
            if nouveau_score > valeur:
                valeur = nouveau_score
                colonne = col
```

Cette logique a été déterminante pour permettre à l'IA de maximiser son score. Nous avons ajusté la valeur en fonction des scores obtenus, s'assurant ainsi que l'IA choisit toujours le meilleur coup possible, en maximisant son score tout en minimisant le score de l'utilisateur.

nous avons utilisé la récursivité pour explorer les différentes possibilités de jeu, tout en ajustant la profondeur à chaque appel. Cela a permis à l'IA de planifier au moins deux coups à l'avance, avec une capacité améliorée par rapport à nos premiers essais où l'IA ne reconnaissait pas toujours les alignements gagnants, en particulier dans les colonnes et les diagonales.
Ainsi, chaque élément de notre implémentation de **minimax** a été soigneusement conçu pour corriger les lacunes observées dans les premières phases de test, conduisant à une IA plus rapide, plus stratégique et plus efficace.

DERNIERS TESTS ET RÉSULTATS

Notre projet a abouti à une IA de Puissance 4 avancée, intégrant les meilleures versions des fonctions ***minimax*** (parmi les trois développées par les membres d'équipe) avec élagage Alpha-Beta et des heuristiques efficaces. Cette IA se distingue par sa capacité à prendre rapidement des décisions stratégiques, améliorant considérablement l'expérience de jeu.

Un progrès notable est la réduction du temps de calcul, rendant les parties contre l'IA plus fluides. La sophistication des heuristiques a également permis à notre IA de s'adapter à diverses stratégies de jeu. Toutefois, nous avons remarqué une baisse de performance lors de l'augmentation de la taille du plateau, un point mis en évidence lors des confrontations avec d'autres IA au sein du groupe de TD, ce qui représente une voie d'amélioration future.

Pour optimiser davantage notre IA, nous envisageons de développer un système calculant le nombre de coups nécessaires pour remporter la partie, afin de privilégier les stratégies gagnantes. De plus, l'élaboration de modèles prédictifs plus complexes pourrait renforcer la capacité de l'IA à prévoir les mouvements des joueurs. Ces améliorations contribueront à faire de notre IA un adversaire encore plus adaptable.

2024

ESILV A3

MERCI