Lina Boulekbache
Benjamin Normand
Julien Fink
Ikhlass Yaya-Oye

TD R - 2020

# Rapport IA : Puissance 4

## Profondeur

La profondeur va nous permettre d'évaluer la valeur d'un état et donner une heuristique. L'idée est la suivante :

La fonction Utility (ou Victoire dans notre code) renvoie 10000 si l'ordinateur gagne et -100000 si ce dernier perd. Au lieu de simplement renvoyer 1, le programme va retourner la valeur de l'utility multipliée par la profondeur.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 4 | 3 |
| 4 | 6 | 8 | 10 | 10 | 10 | 10 | 10 | 10 | 8 | 6 | 4 |
| 5 | 8 | 11 | 13 | 13 | 13 | 13 | 13 | 13 | 11 | 8 | 5 |
| 5 | 8 | 11 | 13 | 13 | 13 | 13 | 13 | 13 | 11 | 8 | 5 |
| 4 | 6 | 8 | 10 | 10 | 10 | 10 | 10 | 10 | 8 | 6 | 4 |
| 3 | 4 | 5 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 4 | 3 |

## Matrice des possibilités/Possibilité

Il s'agit de la matrice indiquant le nombre de possibilités d'aligner 4 pions avec chaque cellule.

On constate que la matrice correspond à 4 fois la partie encadrée en rouge (symétrie à l'horizontale, verticale et diagonale). On a donc calculé un moyen de retomber sur cette matrice peu importe les valeurs qu'on a pour les index lignes et colonnes.

Ainsi après avoir calculé le minimax pour chaque case, on vérifie la valeur dans la matrice des possibilités des cases ayant la valeur d'heuristique la plus élevée. En cas d'égalité, on prend alors l'action qui joue le plus au centre de la grille.

Lina Boulekbache
Benjamin Normand
Julien Fink
Ikhlass Yaya-Oye



# Victoire

Partie ligne

On fait appel à la fonction ligne permettant de déterminer la position du dernier endroit joué. Cela évite de parcourir toute la grille à chaque coup et de gagner en temps.

L'algo regarde la ligne du dernier coup joué pour savoir s'il y a un gagnant.

Partie colonne

L'algo regarde la colonne du dernier coup joué pour savoir s'il y a un gagnant.

Partie diagonale

Pour optimiser le temps de réponse de notre IA, nous allons faire en sorte de ne pas vérifier toutes les diagonales de la grille, mais seulement celles où le dernier joueur vient de jouer car il ne peut y avoir de situation de victoire dans d'autres diagonales. Il y en a donc deux à vérifier la diagonale montante et la diagonale descendante. Pour pouvoir vérifier si celles-ci sont gagnantes nous allons les copier dans des listes et regarder s'il y a 4 éléments identiques de suite.

**Diagonale Montante droite** :

C'est pourquoi nous récupérons le dernier coup joué de coordonnées (ligne, colonne).

On se place sur la case la plus basse de la diagonale :

```
ligneBas, colonneBas = 0, colonne – ligne
if( colonneBas < 0) :
    ligneBas = - colonneBas
    colonneBas = 0
```

On créé une liste qui comportera toute la diagonale montante :

```
diagonale = []
```

Le nombre de cases à vérifier dans la diagonale est :

```
Nb_case = None
If( colonneBas = 0 ): Nb_case = 6 - ligneBas
elif( colonneBas >= 1 and colonneBas <= 6 ): Nb_case = 6
else: Nb_case = 12 – colonneBas
```

Lina Boulekbache
Benjamin Normand
Julien Fink
Ikhlass Yaya-Oye


Les diagonales de la partie jaune ont moins de 4 cases à vérifier. Elles ne peuvent donc pas être gagnantes. On prend seulement en considération celles avec un nombre de cases supérieur ou égale à 4 :

```
if( Nb_case >= 4) :
```

On copie toute la diagonale dans la liste :

```
for i in range (Nb_case):
    diagonale.append(grille[ ligneBas  + i ][ colonneBas + i ])
```

On vérifie s'il y a 4 cases de suite identiques grâce à un compteur (compte) :

```
for j in diagonale:
    if (compte == 4 and precedent != "-") :
        resultat = precedent
        break
    elif (precedent == diagonale [j]): compte += 1
    else :
        precedent = diagonale [j]
        compte = 1
if (compte == 4 and precedent != "-") : resultat = precedent
```

**Diagonale montante droite :**

```
ligneBas, colonneBas = 0, colonne + ligne
if( colonneBas > 11) :
    colonneBas = 11
    ligneBas = ligne – (11 – colonne)
```

| 5,0 | 5,1 | 5,2 | 5,3 | 5,4 | 5,5 | 5,6 | 5,7 | 5,8 | 5,9 | 5,10 | 5,11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4,0 | | | | | | | | 4,8 | | 4,10 | 4,11 |
| 3,0 | 3,1 | | 3,3 | | | | | 3,8 | 3,9 | | 3,11 |
| 2,0 | | | | | 2,5 | | | | | | 2,11 |
| 1,0 | 1,1 | | | h,h | | | | | | | 1,11 |
| 0,0 | 0,1 | 0,2 | 0,3 | 0,4 | 0,5 | 0,6 | 0,7 | 0,8 | 0,9 | 0,10 | 0,11 |

On créé une liste qui comportera toute la diagonale descendante :

```
diagonale = []
```

Le nombre de cases à vérifier dans la diagonale est :

```
Nb_case = None

If( colonneBas == 11 ): Nb_case = 6 - ligneBas

elif( colonneBas >= 5 and colonneBas <= 10 ): Nb_case = 6

else: Nb_case = colonneBas + 1
```

Les diagonales de la partie verte ont moins de 4 cases à vérifier. Elles ne peuvent donc pas être gagnantes. On prend seulement en considération celles avec un nombre de cases supérieur ou égale à 4 :

```
if( Nb_case >= 4) :
```

On copie toute la diagonale dans la liste :

```
for i in range (Nb_case):
    diagonale.append(grille[ ligneBas  + i ][ colonneBas - i ])
```

On vérifie s'il y a 4 cases de suite identiques grâce à un compteur (compte) :

```
for j in diagonale:
    if (compte == 4 and precedent != "-") :
        resultat = precedent
        break
    elif (precedent == diagonale [j]): compte += 1
    else :
        precedent = diagonale [j]
        compte = 1
if (compte == 4 and precedent != "-") : resultat = precedent
```

La fonction Victoire retourne soit 100 000, soit -100 000, soit 0 :

Dans le cas 100 000 : Cela signifie que Victoire a détecté que notre IA a gagné

Dans le cas -100 000 : Cela signifie que Victoire a détecté que l'IA adverse a gagné

Dans le cas 0 : Cela signifie que personne n'a gagné pour le moment. On fait alors appel à la fonction Score_algo qui nous permettra alors de prendre une décision pour le coup suivant (en lien avec Minimax).

Lina Boulekbache
Benjamin Normand
Julien Fink
Ikhlass Yaya-Oye



# Score_algo

Cette fonction prend en paramètres la grille à un état lambda et le coup (différence des valeurs entre un coup de l'adversaire et un coup local).

Elle parcourt une fenêtre de 4 cases (colonnes, lignes, diagonales ; tout confondu) et renvoie un score pour chaque n-quadruplet de cases grâce à la fonction Scoring qui attribue une valeur à chaque quadruplet de cases. Un slicing est effectué pour ne conserver que les 4 cases voulues lors de l'étude d'un état.

Ces différentes valeurs sont sommées pour obtenir la « note » globale de la grille à un état lambda.

La fonction est utilisée dans Minvalue et Maxvalue du Minimax dans le cas où une victoire n'est pas directement détectée en notre faveur en profondeur 3-4 par notre IA.

# Scoring

Voir ci-dessus.

Ces deux méthodes permettent d'avoir une vue globale de notre grille plutôt qu'une vue en un point particulier.

Même si un coup semble le meilleur pour l'IA, il faut également regarder le reste de la grille pour être sûr de ne pas permettre par exemple un piège de l'adversaire. C'est ce que ces fonctions empêchent justement en calculant l'ensemble des coups donnés à une profondeur maximale de 3-4.

De plus, elle permettent aussi un « decision-making » intéressant en début de partie sachant que peu de coups ont été joués et que la prise de décision est délicate dans le sens où l'IA ne peut pas prédire trop de coups à l'avance.

# Possible

Récupère les coups possibles dans une liste et sépare cette liste en deux. La première partie de la liste va être inversée : on parcourt la liste de droite à gauche. On conserve la deuxième liste dans son état d'origine. On crée une troisième liste résultat qui est la combinaison de ces deux premières listes.

Résultat : liste des coups possibles triés du milieu vers les extrémités.

Pourquoi fait-on cela ? : résultat est utilisé dans Maxvalue et Minvalue dans l'élagage alpha-bêta afin de privilégier les coups au centre de la grille (voir matrice des possibilités).

Lina Boulekbache
Benjamin Normand
Julien Fink
Ikhlass Yaya-Oye



# Maxvalue, Minvalue

Fonctions récursives

Maxvalue : On effectue un élagage alpha-bêta afin de ne conserver que les branches les plus pertinentes.

Si jamais la partie est finie ou que la profondeur vaut 0 :

- Si Victoire != 0, on retourne (profondeur+1)*Victoire et on continue l'itération.

Sinon :

- On fait appel à Minvalue

Minvalue : Même procédure que Maxvalue dans le cas contraire…

# Jouer

Jouer prend comme paramètres la grille et la colonne où l'on souhaite jouer. Une copie de cette grille est effectuée à laquelle on rajoute un coup dans cette colonne.

# Minimax

Notre Minimax suit la procédure habituelle du Minimax vue en cours. On récupère le score maximal et si deux coups possèdent ce score , on départage les deux coups grâce à la théorie de la matrice des possibilités (càd jouer plutôt au milieu car davantage de chance de victoires).

Enfin, si deux actions ont la même heuristique et le même nombre de possibilités de gagner, on prend celle plus proche du milieu.

# Exo

La fonction Exo tourne tant que la grille n'a pas de gagnant ou que 42 jetons n'ont pas été posés.

Quand l'IA joue, on fait appel à Minimax qui fait appel à toutes les autres fonctions décrites ci-dessus.

Sinon, c'est à nous de jouer (donc en théorie à l'IA adverse).