Intelligence artificielle TP 2 – Jeu

Antoine Dujardin

II

1:

La profondeur de l'arbre sera toujours égal au nombre de cases restantes.

Si l'arbre était explore après une fin de partie (victoire, défaite, match nul), le nombre de noeuds explorés serait: 9+9*(8+8*(7+7*(...))). Le nombre de noeuds développés est donc inférieur à cette formule.

Cette commande donne le nombre de noeuds créés à chaque tour par `minimax_player`:

```
>print(ttt.play_game(minimax_player, minimax_player))
```

Tour	Noeuds développés
1	549945
2	59704
3	7331
4	934
5	197
6	46
7	13
8	4
9	1

2:

Cette commande donne le nombre de noeuds créés à chaque tour par `alphabeta_player`:

```
>print(ttt.play_game(alphabeta_player, alphabeta_player))
```

Tour	Noeuds développés
1	18296
2	2337
3	843
4	74
5	63
6	16
7	9
8	4
9	1

Alphabeta est 30 fois plus rapide que minmax, et fournit des résultats aussi bons. C'est donc un algorithme supérieur à minmax.

III- Création d’une fonction d’évaluation

1:

Avec la fonction d'évaluation vue en cours (Nb de lignes ouvertes seulement pour la machine – Nbre de lignes ouvertes seulement pour l'adversaire). Le code est dans fEval.py.

Avec cet état du plateau:

```
X O X

O . O

X . .
```

On voit que chaque joueur peut remplir des lignes en jouant sur la case (2, 2). Ils ont tous les deux une ligne incomplète :

```
> total_unfinished_lines_number(my_state.board, 'X')

1

> total_unfinished_lines_number(my_state.board, 'O')

1
```

La fonction `fEval` renvoie donc bien 0 pour les deux joueurs.

```
> fEval(my_state, 'X')

0

> fEval(my_state, 'O')

0
```

Avec cet état:

```
X O X
. . O
X . O
```

Seulement le joueur X peut remplir des lignes.

```
> total_unfinished_lines_number(my_state.board, 'X')
2
> total_unfinished_lines_number(my_state.board, 'O')
0
```

La fonction `fEval` renvoie donc bien 2 pour X, et -2 pour O.

```
> fEval(my_state, 'X')
2
> fEval(my_state, 'O')
-2
```

2:

J'ai utilisé cette fonction pour calculer le taux de victoire d'un joueur contre le joueur aléatoire avec beaucoup d' essais:

```
def test_alphabeta(prof_max, n_tries):
    return statistics.mean([
        ttt.play_game(random_player, alphabeta_player, prof_max=prof_max)
        for i in range(n_tries)
    ]) / -20 + 0.5
```

`ttt.play_game` renvoie 10 si le premier joueur gagne, -10 si le deuxième gagne, 0 sinon.

Avec alphabeta, on a donc:

profondeur: 20, essais: 100, score: 0.915

profondeur: 10, essais: 200, score: 0.9

profondeur: 5, essais: 500, score: 0.935

profondeur: 4, essais: 500, score: 0.911

profondeur: 3, essais: 1000, score: 0.932

profondeur: 2, essais: 1000, score: 0.8815

profondeur: 1, essais: 1000, score: 0.896

profondeur: 0, essais: 1000, score: 0.89

Le score baisse un peu avec une profondeur inférieure à 3, quand l'algorithme ne peut pas réfléchir a plus d'un coup d'avance. Le score reste élevé, ce qui montre que la fonction d'évaluation est assez bonne.

En testant le meme joueur alphabeta contre un autre joueur alphabeta sans limite de profondeur, on a:

profondeur: 20, essais: 100, score: 0.5

profondeur: 10, essais: 100, score: 0.5

profondeur: 5, essais: 100, score: 0.5

profondeur: 4, essais: 100, score: 0.5

profondeur: 3, essais: 100, score: 0.5

profondeur: 2, essais: 100, score: 0.0

profondeur: 1, essais: 100, score: 0.0

profondeur: 0, essais: 100, score: 0.0

Aucun joueur ne gagne la partie quand la profondeur du deuxième joueur est supérieure à 2, après le joueur 1 (sans limite de profondeur) gagne toujours. La fonction d'évaluation est donc bien limitée si la

profondeur est inférieure à 3.

IV – Elagage maximum

```
>print(ttt.play_game(alphabeta_player, alphabeta_player))
```

Tour	Noeuds développés sans ordonnancement	Noeuds développés avec ordonnancement
1	18296	18296
2	2337	2337
3	843	729
4	74	74
5	63	31
6	16	16
7	9	9
8	4	4
9	1	1

L'ordonnancement permet de développer moins de noeuds, mais la différence n'est pas très grande. Une meilleure fonction d'évaluation permettrait d'obtenir de meilleurs résultats.

V – Étendre le jeu à un damier plus grand

La fonction d'évaluation est peu efficace. Elle donne + 1 s'il y a 2 points alignés, et +10 si le jeu est fini. Ce n'est pas assez pour une grille avec une largeur de 4. Il faudrait donner plus d'échelle entre les différents états.

La fonction d'évaluation améliorée donne pour chaque ligne:

État	points
1 point du joueur	+1
2 points du joueur	+10
3 points du joueur	+100
Victoire du joueur	+inf
N'importe quel nombre de points du joueur et de son adversaire	0

Le total est donc : somme des points pour chaque ligne pour le joueur - somme des points pour chaque ligne pour l'autre joueur.