



École Nationale Supérieure de l'Informatique pour l'Industrie et l'Entreprise

Rapport IA IRP

TP Jeux

COCKS Yvon
OMNES Simon

Table des matières

1	Echauffement	2
1.1	Question 1	2
1.2	Question 2	2
1.3	Question 3	2
2	Projet	3
2.1	Jeu de Nim	3
2.1.1	Choix d'implémentation	3
2.1.2	Fonctions particulières : getUtility	3
2.2	Puissance 4	5
2.2.1	Choix d'implémentation	5
2.2.2	Fonctions particulières	5
3	Problèmes rencontrés	6

1 Echauffement

Avant de commencer le projet, voici les trois questions auxquelles nous devons répondre.

1.1 Question 1

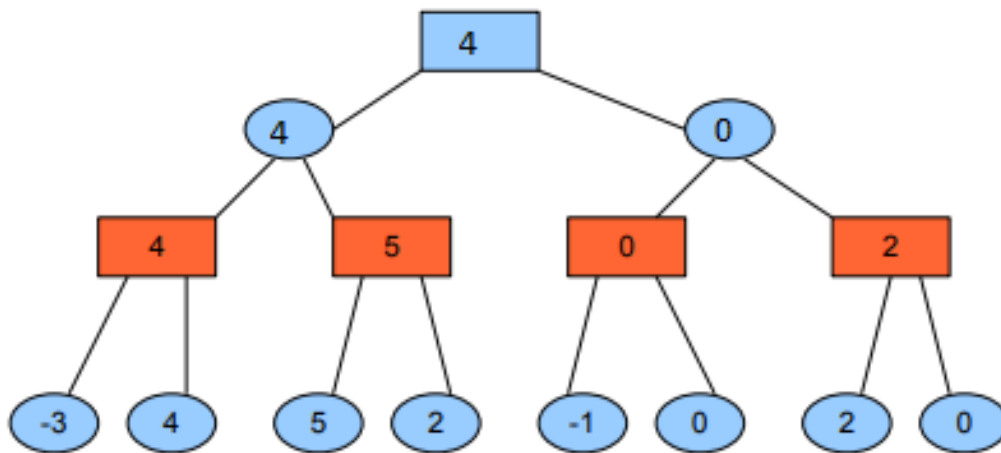


Figure 1: Arbre de décision MiniMax

1.2 Question 2

Le meilleur choix est le choix 1, car le joueur MAX (représenté dans l'arbre par un carré) choisit le score maximal des cases sous lui. AlphaBeta ne remet jamais en question le meilleur choix, car tout ce qui est fait est une suppression des branches qui ne "valent pas le coup" d'être explorées plus loin.

1.3 Question 3

Le carré bleu de gauche avec un 2 sera élagué, car on sait que la position atteinte juste à sa gauche, 5 sera dans tous les cas meilleure que le alpha précédent (qui était de 4). On est donc certain que la branche, même si elle est encore supérieure à 5, n'influera pas sur le choix de coup à l'instant présent. Il est donc raisonnable de l'ignorer pour cette itération.

Le carré orange avec un 2 ainsi que toutes les positions en-dessous seront élagués car le coup 0 mène déjà à une situation pire que le beta de base qui était de 4. Il n'est donc pas la peine d'explorer plus loin cette branche.

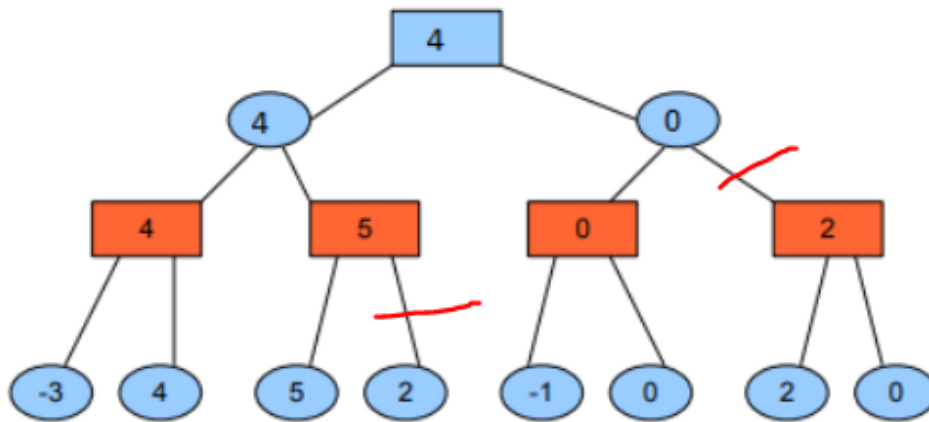


Figure 2: Arbre de décision après l'élagage Alpha-Beta effectué

2 Projet

Dans le cadre de ce projet, nous devons implémenter le jeu de Nim et le Puissance 4.

2.1 Jeu de Nim

On dispose d'un tas d'allumettes (le nombre d'allumettes sera un paramètre du jeu). Deux joueurs choisissent, chacun à leur tour (Max commence) 1, 2 ou 3 allumettes dans la réserve. Le joueur qui retire la dernière allumette de la réserve perd la partie.

2.1.1 Choix d'implémentation

Ici, l'état du jeu est constitué d'un entier qui représente le nombre d'allumettes restantes. Les actions sont représentées par des valeurs entières, c'est-à-dire le nombre d'allumettes retirées par le joueur (1, 2 ou 3).

2.1.2 Fonctions particulières : getUtility

La fonction d'utilité n'est évaluée que lorsque l'état est terminal, i.e qu'il ne reste qu'une allumette. Donc si c'est le cas, on dit que c'est une défaite (-1) si l'ordinateur doit jouer, et une victoire (1) si c'est à l'humain. Si l'état n'est pas terminal, on considèrera qu'aucun des deux joueurs n'a l'avantage (0).

Pour la rendre heuristique, on sait qu'il suffit en fait que le nombre d'allumettes soit congru à 1 modulo 4 pour donner une victoire garantit au joueur dont ce n'est pas le tour. Cela a réduit le nombre de noeuds cherchés drastiquement (on parle de 500000 sans la profondeur

limitée à quelques centaines avec).

Pour l'élagage, on rajoute `fold_until_alpha` et `fold_until_beta` des folds spéciaux pour quitter l'itération dès que `value` a une certaine valeur, et on ajoute des variables impératives pour pouvoir changer les valeurs de `alpha` et `beta`. On a implémenté directement l'élagage alpha-beta avec profondeur limitée, car il suffit d'enlever la condition `d != 0` dans le code pour passer à une profondeur illimitée.

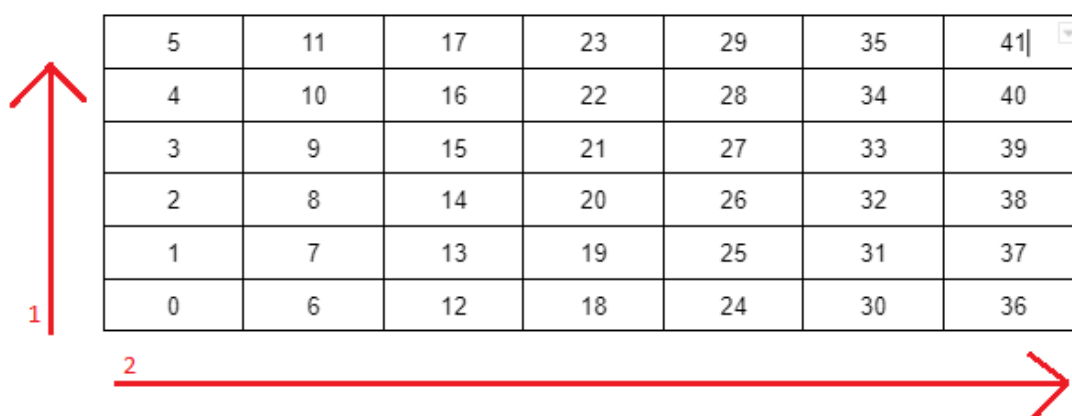
L'ordinateur ainsi créé ne me bat cependant pas à tous les coups, car le jeu des allumettes est facilement résolu et il est courant de se retrouver dans une situation où la victoire est garantie, et ce sans avoir besoin de capacités de calcul inhumains.

2.2 Puissance 4

Sur une grille de 7 colonnes de 6 cases, chaque joueur met à tour de rôle un pion dans une colonne, celui-ci descend en bas de la colonne. Celui qui en aligne 4 a gagné.

2.2.1 Choix d'implémentation

L'état est représenté pour une liste d'entier de taille 42 (7*6). Comme sur l'image ci-contre :



5	11	17	23	29	35	41
4	10	16	22	28	34	40
3	9	15	21	27	33	39
2	8	14	20	26	32	38
1	7	13	19	25	31	37
0	6	12	18	24	30	36

Figure 3: Représentation du Puissance 4

Les actions possibles sont les colonnes, la première étant la colonne 0 et la dernière la colonne 6, sur lesquelles le joueur peut placer un jeton.

2.2.2 Fonctions particulières

Nous avons créé plusieurs fonctions annexes pour cette partie.

Tout d'abord, la fonction **first_avail** qui nous renvoie la position de la case vide sur laquelle le jeton "tombera" permettant ainsi de passer un chiffre représentant une action comprise en 0 et 6 à un entier compris entre 0 et 41.

$$val \text{ first_avail} : state \rightarrow action \rightarrow int = \langle fun \rangle$$

Ensuite, les fonctions **check(Direction)4** et **count(Direction)(X)** avec **Direction** $\in \{Horizontal, Vertical, DiagUp, DiagDown\}$ et **X** = 2, 3 ou 4

Les fonctions **check(Direction)4** qui regardent s'il y a 4 jetons alignés de même valeur sur les lignes, colonnes ou diagonales, et renvoient un couple de booléens. Le premier élément du couple correspond à l'ordinateur et le second au joueur.

$$val \text{ check}(\text{Direction})4 : state \rightarrow (bool * bool) = \langle fun \rangle$$

Les fonctions **count**(*Direction*)(*X*) qui comptent le nombre d'alignement de 2, 3 ou 4 jetons alignés de même valeur sur les lignes, colonnes et diagonales, et renvoient le résultat sous forme d'un couple d'entiers. Le premier élément du couple correspond au nombre d'alignement de 2, 3 ou 4 jeton de l'ordinateur et le second à ceux du joueur.

$$val count(Direction)(X) : state \rightarrow (int * int) = \langle fun \rangle$$

Enfin, la fonction **getUtility** évalue via une combinaison linéaire du nombre d'alignement de 2,3 ou 4 jetons multiplié par différents scalaires qui note leur importance dans le jeu, l'alignement de 4 jetons ayant le plus gros scalaire :

$$((20Nb4P1 + 5Nb3P1 + 2Nb2P1) - (20Nb4P2 + 5Nb3P2 + 2Nb2P2))$$

avec Nb4P1 qui correspond au nombre d'alignement de 4 de l'ordinateur, Nb3P1, au nombre d'alignement de 3 de l'ordinateur et Nb2P1 au nombre d'alignement de 2 de l'ordinateur. Et avec Nb4P2 qui correspond au nombre d'alignement de 4 du joueur, Nb3P2, au nombre d'alignement de 3 du joueur et Nb2P2 au nombre d'alignement de 2 du joueur.

3 Problèmes rencontrés

Lors du premier lancement de puissance 4, nous avons eu un problème de boucle infini. Nous avons donc utilisé `ocamlc -g`, `ocamlrun` et des `printf` pour remonter à la source du problème.