Amadéo GILLIERON, Felicity DORLEANS, Géraud FELZINES, Thessa ERIC

Projet d'Intelligence Artificielle : GOMOKU

Nos choix et implémentations de la structure de données

QUELQUES DEFINITIONS

Afin de construire notre algorithme, nous avons eu besoin de percevoir l'état du jeu à chaque nouvelle évolution (à chaque nouveau coup joué). Pour pouvoir faire cela, nous avons introduit la notion de barre de 5 cases. Dans les prochaines lignes, nous allons davantage définir ces barres, ainsi que d'autres notions.

Barre de 5 cases (barres) :

Une barre est une suite de 5 cases pouvant représenter un alignement de 5 jetons gagnants. Par exemple :

$A1 - A2 - A3 - A4 - A5$ (En bleu)

$B2 - C3 - D4 - E5 - F6$ (En jaune)

$C1 - D1 - E1 - F1 - G1$ (En vert)

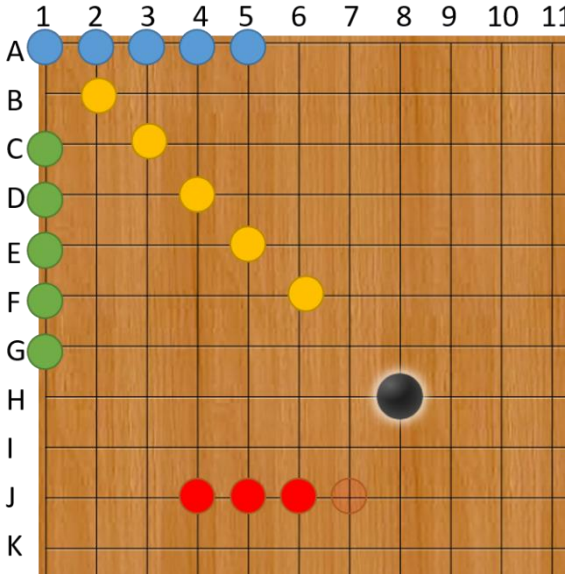

Figure 1 : Explication de la notion de barres

Coup gagnant :

Un coup gagnant est un coup où l'adversaire perdra à notre tour d'après. Par exemple, dans la *figure* 1, au niveau des pions rouges, si c'est à rouge de jouer, un coup gagnant sera en J7 ou en J3, car peu importe ce que fera l'adversaire, (à part s'il pouvait gagner en un coup), rouge gagnera lorsque ça sera son tour.

Etat du jeu :

Notre algorithme fonctionne beaucoup avec des sets, ce qui nous permet de trouver des éléments extrêmement vite. Nous avons représenté l'état du jeu par plusieurs ensembles de barres, stockés dans des sets. Ces sets ont chacun un usage spécifique. Certains de ces sets sont rassemblés dans des listes. Pour compléter cette représentation, nous utilisons aussi un set de cases déjà jouées. Vous pouvez voir sur la *figure* 2 une représentation imagée, et légendée plus bas :

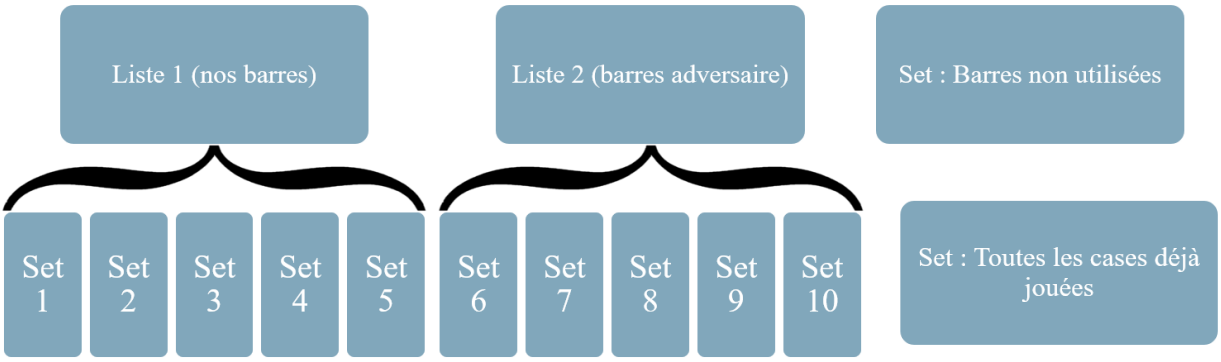

Figure 2 : Notre représentation de l'état du jeu

- Liste 1 : Nous y trouvons les ensembles de barres contenant seulement nos jetons. Il y a cinq sets. Le set 1 contient les barres où sont présents 5 de nos jetons (toute la barre est composée de nos jetons en fait), les barres du set 2 en contiennent 4, celles du set 3 en contiennent 3, celle du set 4 en contiennent 2, et les barres du set 5 en contiennent seulement 1.
- Liste 2 : Nous y trouvons les ensembles de barres contenant seulement les jetons de l'adversaire. Il s'agit du même raisonnement que pour la liste 1.
- L'état du jeu est aussi représenté par un ensemble de barres non utilisées par aucun des deux joueurs, donc ne contenant aucun jeton du tout au début de la partie.
- Finalement, on représente aussi l'ensemble des cases déjà jouées.

Si une barre contient 2 jetons de couleurs différentes alors elle n'est plus prise en compte et a été effacé (discard) du set où elle était présente. En effet, il est impossible qu'elle soit alors complétée par personne, vu qu'un jeton adverse se trouve dessus.

Par construction de cet état du jeu, une barre ne peut être présente que dans un seul des ensembles de barres présentés ci-dessus à la fois.

Dans le code, les Liste 1, Liste 2 et le set des barres non utilisées forment une liste de 3 éléments (Liste 3). Au final, un état du jeu est donc un tuple formé de Liste 3 et du set des cases déjà jouées.

LES CHOIX ET IMPLEMENTATIONS DES FONCTIONS DU MINMAX

Nous allons maintenant aborder les fonctions utiles au minimax, en détaillant au maximum ces choix et pourquoi nous en avons décidé ainsi.

Fonction actions :

Elle prend en paramètres d'entrée un état comme décrit précédemment (voir plus haut).

Pour optimiser la vitesse de la fonction et la place des return, la fonction est divisée en deux parties.

Comme il est décrit dans l'article intitulé « Go-Moku and Threat-Space Search » mis à disposition dans le sujet, page 4, paragraphe 3.3, seulement un nombre limité des meilleures actions est considérée pour un coup dans l'alpha-beta search. Notre fonction actions s'efforce donc de trouver les actions (cases) faisant partie d'une séquence gagnante ou qui présente un intérêt stratégique. Pour cela, nous nous sommes appuyés sur les différents diagrammes de l'article. Le but étant de repérer, grâce aux barres représentant un état, ces meilleures cases.

L'utilisation des barres permet de détecter les menaces à temps, et de réagir. Dans les prochaines lignes illustrant ces menaces, les captures d'écran sont extraites de l'article susdit afin d'étayer notre argumentaire.

Quelques exemples :

- Un coup gagnant serait de créer 2 menaces de type 1d (1d est la référence de l'article évoquée plus haut, voir *figure* 3 dans ce rapport), ainsi le défenseur ne peut pas défendre les 4 extrémités crées en même temps.

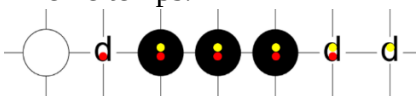

Figure 3 : Menace de type 1d

Sur la *figure* 3, on peut donc placer deux barres (ici en jaune et rouge) provenant strictement du set 3 ou du set 8 (voir *figure* 2). C'est-à-dire, 2 barres contenant 3 jetons du même joueur, et ayant une intersection jouable non nulle (le deuxième d en partant de la droite représente un coup gagnant, sauf si l'adversaire a déjà un coup gagnant). Ce raisonnement en barre est réplicable aussi lorsque les barres ne proviennent pas des mêmes 3 jetons alignés par exemple (Ex : si nous retrouvons deux menaces de type 1a au minimum, voir *figure* 4).

- Un autre coup gagnant serait de créer une menace de type 1a (*figure* 4) et de type 1d (*figure* 3) en même temps, ce qui suit la même logique avec 3 extrémités à défendre en même temps.

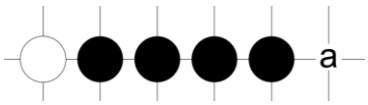

Figure 4 : Menace de type 1a

- Des coups intéressants seraient par exemple le croisement de 2 paires de jetons alignés (Diagram 6a, au coup 29 dans l'article). Ou bien l'alignement de 4 pions sur une barre de 5, ce qui créé une obligation de jouer sauf si l'adversaire peut déjà gagner en 1 coup (Diagram 6d, au coup 25 dans l'article, voir *figure* 5 dans ce rapport).

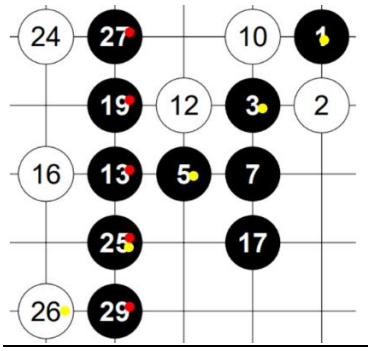

Figure 5 : Menace par alignement de 4 pions sur une barre de 5

Sur cette dernière *figure*, Nous voyons que le coup 25 est gagnant, car il est à l'intersection d'une barre de 3 (coups 1, 3 et 5) et d'une barre de 2 (coups 13 et 19), ce qui nous permet de retrouver une menace de type 1a et de type 1d assurant une victoire. Ce type de coup parmi d'autres est toujours détecté par la fonction *actions()*.

Fonction résultat (état, action) :

Elle prend en paramètres d'entrée un état et une action c'est-à-dire une case donnée, ainsi que le joueur jouant la case (nous ou l'adversaire).

Par construction des barres, une case appartient à plusieurs barres (par exemple A6 appartient à 5 barres sur la ligne A, 2 barres diagonales et 1 barre sur la colonne 6).

Lorsqu'une case est jouée par l'un des deux joueurs, les barres appartenant au joueur adverse contenant cette case sont effacées des sets de l'adversaire car on n'aura plus l'utilité de ces barres (elles ne pourront jamais être complétées par des jetons de même couleur car étant elles même déjà composées de deux jetons de couleurs différentes).

Tandis que les barres appartenant au joueur jouant la case, donc contenant la case jouée, changent de set (elles se décalent vers les sets les plus à gauche sur le schéma car elles contiennent un jeton supplémentaire du joueur). Les barres qui n'avaient pas encore été utilisées et qui contiennent la case jouée sont alors déplacées vers le set 5 ou 10 selon le joueur, car elles contiennent maintenant le jeton du joueur jouant la case.

La case est ajoutée au set des cases jouées.

La fonction renvoie les 2 listes de set, la nouvelle Liste 1 et la nouvelle Liste 2, ainsi que les 2 autres sets qui ont été modifiés. (cf. *figure* 6)

En résumé on effectue une redistribution des barres à chaque coup et donc, selon la conception de notre algorithme, une actualisation des états.

Nous pouvons voir un exemple ci-dessous :

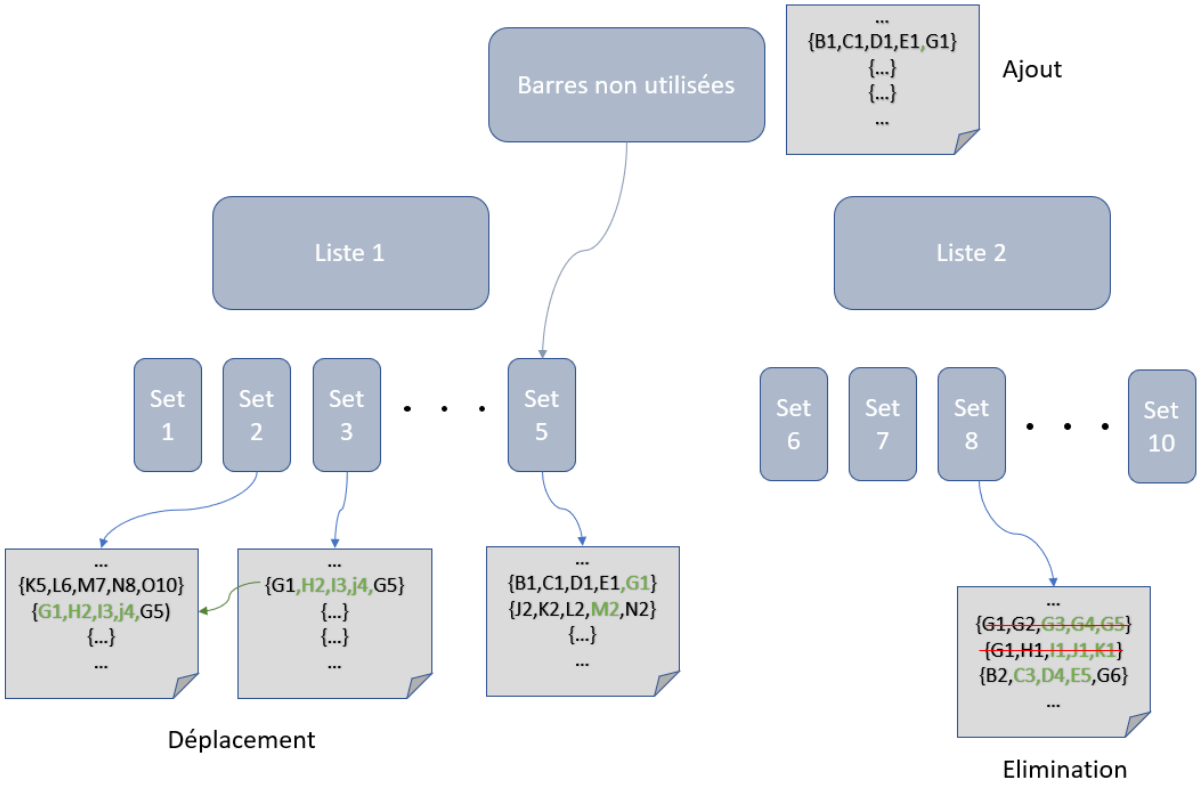

Figure 6 : Schéma des états – exemple si le joueur de la Liste 1 joue en G1

Fonction Terminal-test (état) :

L'état en paramètre d'entrée est terminal si et seulement si l'un des 2 joueurs a une barre contenant 5 jetons joués, c'est-à-dire que l'un des set 1 (ensemble des barres contenant exactement 5 de nos jetons) ou set 6 (ensemble des barres contenant exactement 5 des jetons adverses) (cf. *figure 6*) est non vide, ou bien que 120 jetons aient déjà été posé sur le plateau, c'est-à-dire que le set des cases déjà jouées est de longueur 120.

Fonction Utility (Liste1, Liste2) : (notée fitness dans le code)

Elle prend en paramètre d'entrée une partie de la représentation d'un état, elle n'a seulement besoin que de Liste 1 (nos barres) et de Liste 2 (barres adverses).

Des poids par ordre croissant sont associés respectivement aux barres contenant strictement 1, 2, 3, 4 et 5 jetons d'un même joueur. Le poids des barres adversaires est compté négativement tandis que le poids de nos barres est compté positivement. On accorde un poids plus important aux barres contenant un nombre important de jetons. Donc plus un joueur a de barres importantes, en tenant compte de la pondération, et plus la fonction de fitness tendra à dire que le joueur est avantagé dans cet état présent.

Nous souhaitons donc ici maximiser cette fonction.

Fonction minimax avec élagage alpha-beta :

Pour implémenter cette fonction, nous avons suivi le pseudo-code indiqué dans le TD4 « IA4-Minimax ». Concernant la question 5, notre $ALPHA\_BETA\_SEARCH$ ne s'arrête pas à une profondeur stricte, mais peut s'enfoncer plus profondément dans l'arbre des actions selon la taille des branches : c'est-à-dire si nous nous trouvons dans une « *threat sequence* », c'est-à-dire où le choix des cases à jouer est mince (obligation de jouer à certains endroits pour ne pas perdre) alors on se permet d'aller étudier à une profondeur supérieure, tandis que sur une séquence d'actions ne présentant pas ce genre de coups, la recherche s'arrêtera plus tôt en profondeur.

Ainsi cela permet à notre fonction $ALPHA\_BETA\_SEARCH$ de prévoir certaines séquences d'actions avec **une profondeur maximale de 12 coups** menant soit à une victoire, soit à une défaite (en limitant la période de calcul à 5 secondes imposées). En revanche avec une limite de 20 secondes, nous battons (en tant que white et en tant que black) en difficulté hard l'algorithme du lien internet d'entrainement. Le paramètre de limitation n'est donc pas une profondeur mais une limite de feuilles de l'arbre des actions variant de $2.500.000$ à $12.500.000$ selon la forme de l'arbre. Seulement environ $\frac{1}{40}^{\text{ème}}$ des états des feuilles sont calculés grâce à l'élagage alpha-beta.

La fonction fitness varie principalement dans l'intervalle $[-300\,;300]$, donc nous avons imposé une valeur de ± 10.000 fixe lorsque l'IA ou son adversaire gagne la partie à un état donné. Cela permet d'éviter toute confusion entre une victoire/défaite et une fonction fitness haute ou basse. Donc si une des premières actions à la racine de l'arbre renvoie une valeur de $10.000 > 8.000$, c'est-à-dire que c'est un coup menant vers une « *winning threat sequence* » alors il est joué directement sans attendre le reste du calcul de l'arbre (inutile car une victoire est déjà calculée et sûre d'être atteinte, sauf erreur de la fonction actions sur une « *threat sequence* », ce qui n'est jamais arrivé sur la dernière version testée plus de 50 fois sur internet).

Fonctionnement :

Une fonction affichage a été implémentée pour une meilleure lisibilité de la situation sur le plateau de jeu, ainsi que l'utilisation d'un module de couleur (noir/jaune, le blanc étant confondu avec le plateau sur la version Jupyter, blanc et rouge sur la version python).

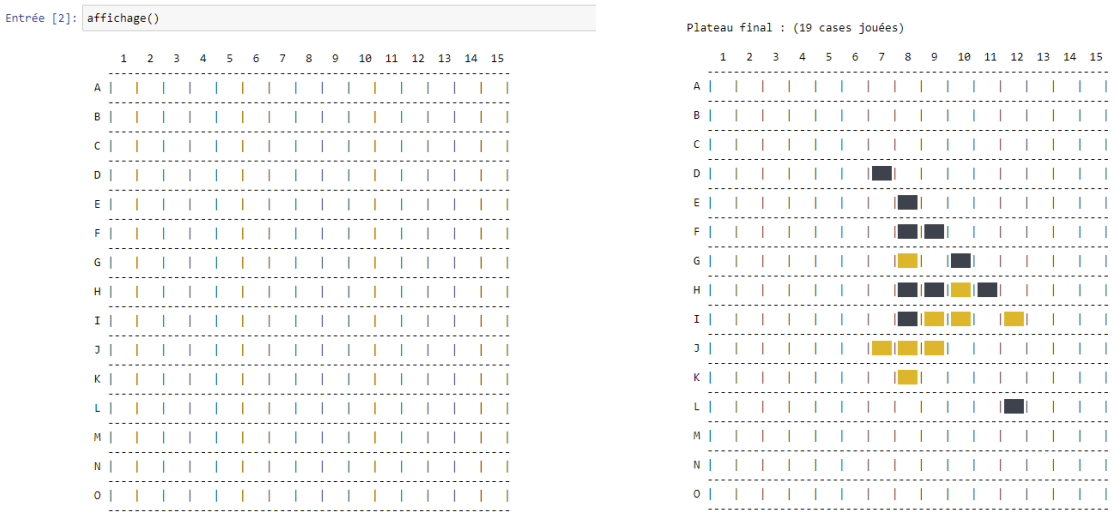

Figure 7 : L'affichage

Pour faire des parties avec notre code, il suffit d'appeler la fonction *jeu()* sans aucun paramètre. Un input demandera qui commence en *H*8, il faudra répondre 0 si l'IA commence en tant que noir, sinon 1 pour commencer soit même/joueur extérieur en tant que noir en *H*8. Puis, il n'y a qu'à suivre l'évolution du plateau et insérer les cases jouées contre l'IA, directement avec leur nom. Par exemple pour jouer la case I9, il suffit de taper I9 lorsque l'IA demande le coup adverse (la saisie est sécurisée en cas de faute de frappe, ou de cases erronées/déjà jouées). C'est-à-dire que la case sera alors redemandée.