

UNIVERSITÉ DE GENÈVE

TRAVAIL DE BACHELOR

Author: Joao Filipe Costa da Quinta

E-mail: Joao.Costa@etu.unige.ch

Accompagnateur: Stéphane Marchand-Maillet

2020 - 2021



**UNIVERSITÉ
DE GENÈVE**

FACULTÉ DES SCIENCES
Département d'informatique

Introduction

Dans le monde de l'informatique nous sommes en contact avec plusieurs algorithmes différents, un algorithme n'est rien d'autre qu'une recette de règles/instructions qui nous permet, en quelque sorte, de résoudre un problème. Il y a des algorithmes qui sont assez simples et logiques, par exemple un algorithme type Greedy (gluton), alors que d'autres seront bien plus complexes, le plus on rentre dans le monde de l'intelligence artificielle le plus ils seront complexes. Personnellement, j'ai toujours mieux compris un algorithme après avoir vu une petite visualisation qui démontre le fonctionnement de l'algorithme.

Dans mon projet, je vais développer des outils de compréhension pour un algorithme, le minimax, c'est l'un des premiers qu'on voit dans le domaine de l'intelligence artificielle, ou la théorie du jeu, c'est un algorithme qui dans sa base est très simple, mais qui devient bien plus complexe quand on tente de l'optimiser, pour le faire je vais mettre en place un jeu qui respecte les critères de l'algorithme, qui pourra être joué par deux entités différentes qui utilisent l'algorithme minimax à des profondeurs de recherche différentes, pour regarder l'efficacité de l'élagage alpha-bêta j'afficherai aussi la quantité de noeuds ignorés. Je développerai aussi un petit outil où l'utilisateur pourra donner comme input tout arbre de recherche souhaité, pour voir le résultat de la recherche minimax.

État de l'art

Il existe plein de sites où l'on peut trouver une explication de l'algorithme minimax avec un petit arbre d'exploration à côté, ainsi que des explorations interactives d'un arbre simple. Par contre, à la date de début du projet, je n'ai pas trouvé de ressources de visualisation pour un grand arbre d'exploration minimax. Si on décide d'appliquer l'algorithme minimax, ce ne sera pas pour un simple arbre avec une dizaine de noeuds, ça sera pour un grand arbre avec des milliers de noeuds, c'est ce point là que je cherche à explorer dans mon projet.

Minimax

Le minimax est un algorithme très sympathique qu'on voit dans la théorie du jeu, il peut être appliqué dans tout jeu où un joueur A affronte le joueur B, les deux jouent en alternance et où la victoire du joueur A implique la défaite du joueur B. L'algorithme part du principe que les deux joueurs feront toujours le meilleur choix possible. Le jeu en question doit être à somme nulle, un score positif implique que le joueur A est gagnant, réciproquement un score négatif implique que le joueur B est gagnant, alors qu'un score nul décrit un état d'égalité. Le morpion, les échecs et les dames sont des exemples de jeux auxquels on peut appliquer cet algorithme du minimax. L'idée de l'algorithme est très simple, le joueur A regarde toutes les configurations de jeu que l'on peut atteindre depuis un état de jeu donné en respectant les règles de transition spécifiques au jeu. Puis, pour toutes les configurations de jeu trouvées, il refait la même chose, mais en se mettant à la place du joueur B. Une fois qu'on arrive à la limite du jeu, A peut regarder lequel des états finaux lui convient le plus et qui est en même temps le plus probable. Il pourra ainsi retracer les étapes qui mènent à ce même état de jeu et faire le choix qui lui donnera de l'avantage face à B.

Pour savoir si une configuration de jeu avantage plus le joueur A ou le joueur B, il faut avoir un moyen de comptabiliser cette configuration de jeu, cette tâche sera accomplie par la fonction d'évaluation. Une fonction d'évaluation doit être capable d'analyser toute configuration de jeu atteignable et retourner un chiffre qui indique quel joueur est avantageux.

Dans tous les jeux plusieurs fonctions d'évaluation sont possibles, par exemple chez le jeu de dames on peut créer une fonction d'évaluation qui retourne $+1/0/-1$, selon quel joueur possède plus de jetons, $+1$ indiquerait que le joueur A possède plus de jetons, 0 indiquerait que les deux joueurs possèdent le même nombre de jetons, finalement, -1 indiquerait que le joueur B possède plus de jetons.

Regardons l'exemple suivant simple (Figure 1), la couleur verte symbolise un état (configuration de jeu) où il est au joueur **A** de jouer, donc il cherche à **maximiser** le score. La couleur rouge symbolise un état où il est au joueur **B** de jouer, donc il cherche à **minimiser** le score. Les chiffres dans les états correspondent aux valeurs retournées par la fonction d'évaluation, on remarque que la fonction d'évaluation est appliquée uniquement aux états finaux. Les chiffres en bleu le nom de l'état. 0

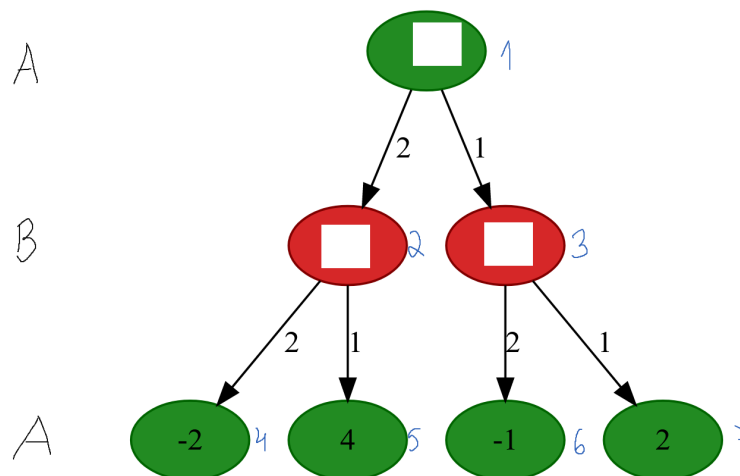


Figure 1: Exemple simple: partie 1

Le joueur A étant à l'état 1, sait qu'il peut atteindre l'état 2 avec la transition 2, ou alors l'état 3 avec la transition 1, depuis l'état 2, le joueur B peut atteindre l'état 4 et 5 avec la transition 2 et 1 respectivement, la même logique est appliquée pour l'état 3. Le but du joueur A est de maximiser la valeur finale, alors que celui de B est de la minimiser.

Si le joueur A regarde que les états finaux, il voit que le score après 2 coups aura 4 possibilités différentes, $[-2, 4, -1, 2]$, vu que le joueur A veut maximiser le score, il décide qu'il veut atteindre l'état 5 qui vaut 4. Pour le faire il faut choisir la transition 2 depuis l'état 1, qui nous mène à l'état 2. Par contre, le joueur B

étant lui même un bon joueur, se trouvant à l'état 2, va plutôt choisir la transition 2, qui résulte dans un score de -2, plutôt que la transition 1 que A souhaitait.

L'algorithme minimax pourra éviter cette situation (Figure 2), il prend en compte le fait que si le joueur B se retrouve dans l'état 2, il va jouer la transition 2 car il cherche à minimiser le score final. Ce qui veut dire qu'on peut quantifier le score des états non finaux 2 et 3, en réfléchissant à quel coup B choisirait.

En effet: $\text{minimax}(\text{état 1}) = \text{maximum}(\text{minimax}(\text{état 2}), \text{minimax}(\text{état 3}))$, l'algorithme est récursif.

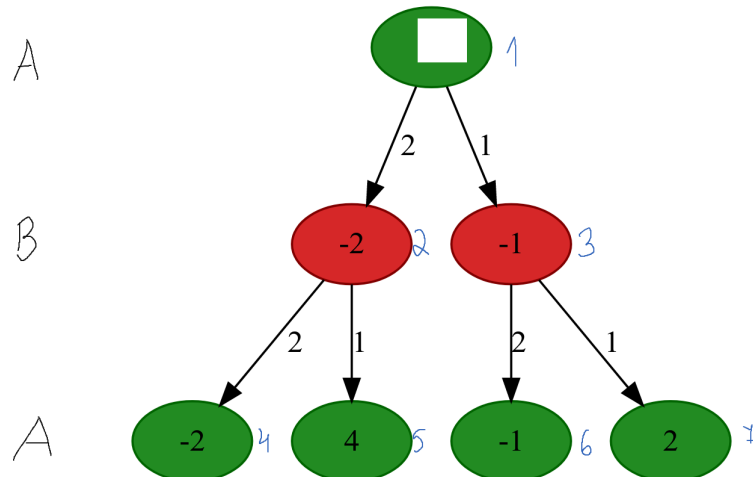


Figure 2: Exemple simple: partie 2

En effet, A peut partir du principe que l'état 2 'vaut' -2 car si A donne l'opportunité à B il choisira le coup qui mène le jeu à l'état 4. En appliquant la même réflexion à l'état 3, on voit qu'il 'vaut' -1, ce qui veut dire que depuis l'état 1, le meilleur score possible après deux coups sera -1, car A cherche à maximiser le score final, on peut dire que A cherche le Max du Min. Les états finaux ont un score calculé à l'aide de la fonction d'évaluation, alors que les états non finaux sont le Min ou le Max des configurations atteignables depuis eux mêmes.

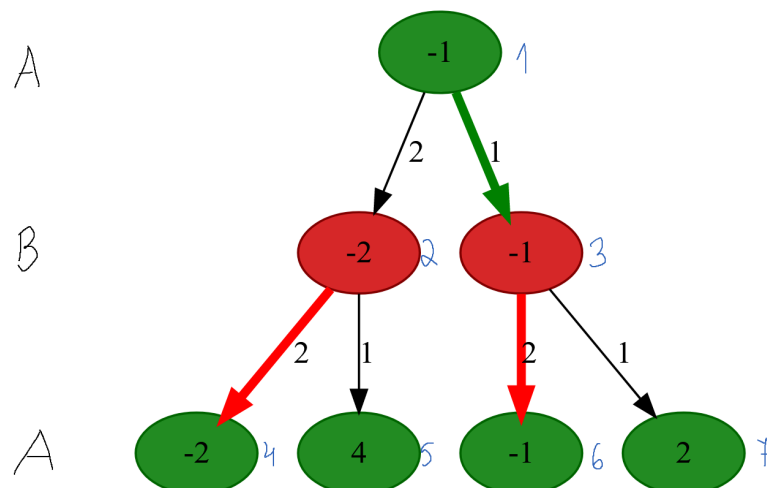


Figure 3: Exemple simple: partie finale

Voici l'arbre d'exploration minimax de cet exemple simple (Figure 3). Les couleurs des transitions symbolisent depuis quel noeud vient sa valeur, il suffit donc de 'suivre' les flèches rouges et vertes jusqu'à l'état final. Dans l'exemple on comprend que A choisit la transition 1, pour atteindre l'état final 6, qui est le maximum du minimum. On comprend aussi le nom de l'algorithme, il explore l'arbre en choisissant le maximum et le minimum en alternance qui correspondent aux coups du joueur A et du joueur B. Il est très simple à

comprendre et à utiliser. En théorie on peut l'appliquer à tout jeu compatible, et développer l'entièreté de l'arbre d'exploration depuis l'état initial, puis faire le meilleur coup possible dès le début.

Cette technique peut être appliquée à un jeu comme le morpion où l'arbre d'exploration est assez petit, on a moins de $9! = 362'880$ noeuds terminaux. Le même ne peut pas être dit pour un jeu un peu plus complexe comme les dames ou les échecs. Chez les échecs on a un facteur de branchement égal à 35, et une partie aura en moyenne 100 coups joués (50 par joueur), ce qui résulte en $35^{100} = 10^{154}$ états, ce qui rend le minimax non utilisable.

En pratique, on va plutôt limiter notre recherche à une profondeur d'arbre donnée, le moins profond, le moins complexe notre calcul sera, mais en revanche notre coup sera probablement moins bon.

Soit b le facteur de branchement moyen, c'est à dire que pour tout état donné x on pourra en moyenne atteindre b états différents. Soit m la profondeur d'arbre que l'on souhaite atteindre, le nombre d'états dans notre arbre sera alors bornée par b^m . L'algorithme minimax réalise une recherche en profondeur (DFS), alors sa complexité en temps est $O(b^m)$ et en mémoire $O(b * m)$. Pour continuer avec l'exemple des échecs, un joueur débutant réfléchi à $m = 4$, on a donc $35^4 = 1'500'625$ états différents. Ce qui est inférieur à 10^{154} le chiffre mentionné ci-dessus. Cela représente encore un très grand nombre d'états pour un ordinateur, on va donc essayer d'optimiser l'exploration.

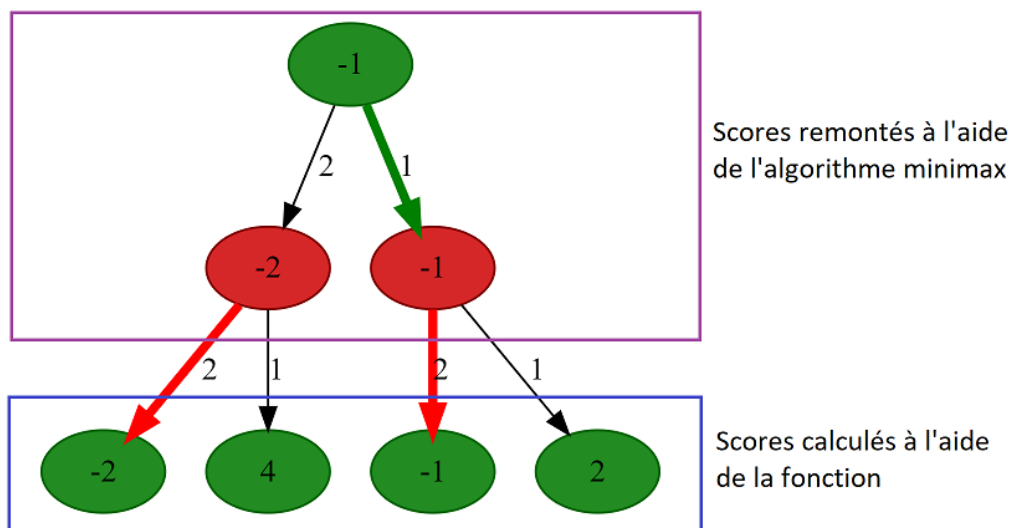


Figure 4: Origine des scores

Ce qu'il faut retenir, c'est que les scores des noeuds finaux sont calculés à l'aide de la fonction d'évaluation. Alors que les scores des noeuds non terminaux, qui ont des enfants, sont remontés grâce à l'algorithme minimax.

Minimax: Élagage alpha-bêta

L'exploration d'un arbre minimax est exponentielle, on ne pourra pas éliminer complètement la difficulté exponentielle, mais on peut tout de même essayer de la réduire. L'élagage alpha-bêta est une variante très élégante du minimax qui nous permettra d'ignorer certains états de jeu, tout en trouvant la même solution que le minimax qu'on a vu.

Analysons l'exemple suivant (Figure 5), c'est le graphique de l'exploration d'un arbre minimax avec l'élagage alpha-bêta. Comme avant, les noeuds verts et rouges indiquent que c'est au joueur A ou B de jouer respectivement. Les chiffres dans les états finaux correspondent à leur score exacte, alors que les chiffres dans les états non finaux, correspondent au max/min de leurs enfants, c'est-à-dire les états que l'on peut atteindre. Les transitions vertes et rouges symbolisent de quel état vient la valeur. Les chiffres en bleu montrent dans quel ordre les noeuds ont été visités par l'algorithme. Dans la suite, on verra en plus de détail quel est l'importance de l'ordre dans laquelle les noeuds sont visités.

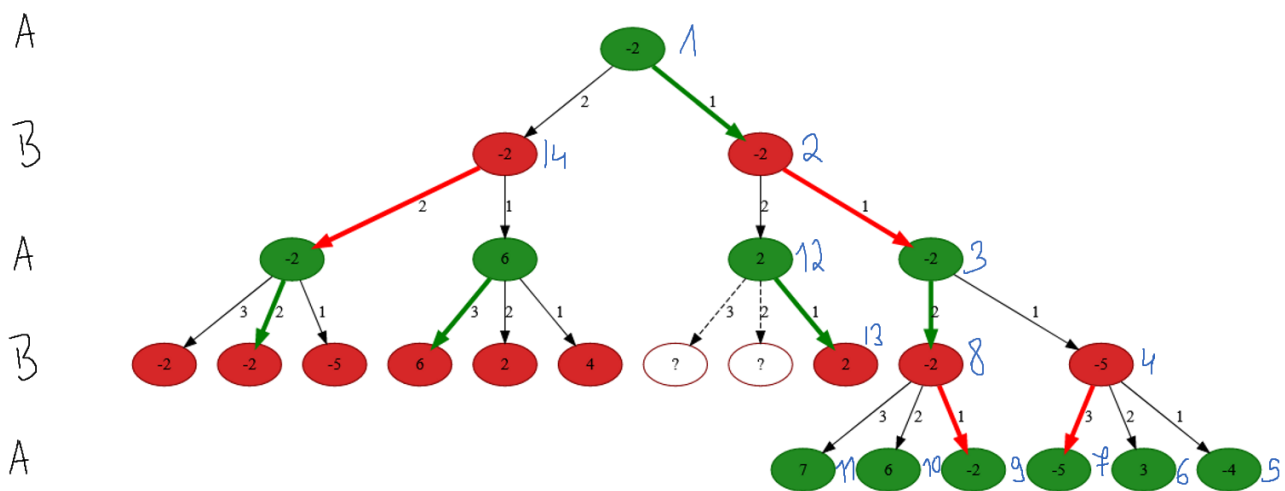


Figure 5: Élagage alpha-bêta

Dans cet exemple l'algorithme avec élagage se comporte exactement comme l'algorithme sans élagage jusqu'à ce qu'on arrive au noeud 12. Faisons une récapitulation de la situation lorsqu'on arrive l'état 12. Le joueur B est à l'état 2 et à 2 possibilités. La première qui est déjà explorée, mène à l'état 3 qui 'vaut' -2. Avant d'explorer la deuxième option, la première garantie déjà un score de -2. En commençant l'exploration de la deuxième possibilité, on arrive à l'état 12, et on arrive à l'état final 13 qui vaut 2. Le joueur B peut maintenant s'arrêter. En effet, si le joueur B choisit la deuxième option il sait déjà que le joueur A à son tour pourra choisir d'aller dans l'état 13, qui vaut 2. Comme 2 est supérieur à -2 il n'a pas besoin de visiter les autres enfants du noeud 12. Car il ne veut pas donner au joueur A, qui est un maximum, la possibilité de faire le coup qui mène à l'état 13, qui vaut 2.

En effet, $\text{minimax}(\text{etat } 12) = \text{maximum}(2, ?, ?)$ qui vaut au moins 2.

Il y a 2 noeuds qui sont complètement ignorés par l'algorithme, et le fait qu'on les ignore ne change pas le choix final, d'où l'élégance de l'algorithme alpha-bêta. Ce qui est le plus intéressants c'est que si les 2 noeuds ignorés avaient des enfants, on aurait pu les ignorer ici. Du coup, théoriquement dans un arbre plus profond, avec un branchement plus important, on pourra ignorer de plus en plus des noeuds.

Regardons encore un exemple (Figure 6), qui nous fera comprendre l'importance de l'ordre.

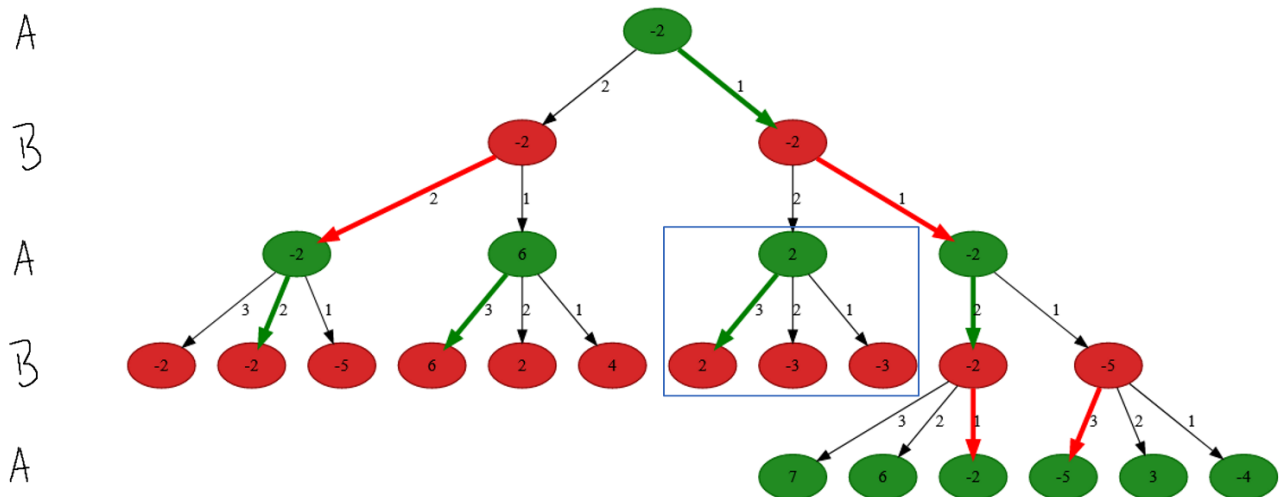


Figure 6: Élagage alpha-bêta 2

En regardant la figure 6, on remarque une ressemblance avec l'exemple précédent. La seule différence c'est qu'ici on n'a pas pu ignorer des noeuds, mais l'algorithme reste le même. L'arbre est aussi presque le même qu'avant, la différence est l'ordre dans lequel l'algorithme visite les noeuds encadrés. En réalité, la variante minimax avec l'élagage alpha-bêta a exactement la même complexité en temps et mémoire que le minimax simple, le seul avantage est que la version améliorée, peut avoir une chance d'ignorer quelques noeuds. Le problème est que ça ne dépend pas de l'algorithme lui-même, mais uniquement de l'ordre dans lequel les enfants de chaque noeud sont organisés.

En théorie, il est possible d'ordonner les coups de façon à maximiser les noeuds élagués (Figure 7), alors on pourrait améliorer la complexité du minimax simple, on passerait d'une complexité en temps de $O(b^m)$ à $O(b^{m/2})$. Ce qui correspond à avoir un branchement \sqrt{b} . Si on reprend l'exemple des échecs où on avait $b = 35$ et $m = 4$, avec un ordre de coups optimal, on retrouverait un facteur de branchement d'environ $b = 6$. Ce qui veut dire qu'on peut résoudre le même problème, si m reste le même, en la moitié du temps. Ou alors on peut faire une exploration de profondeur $2 * m = 8$ dans la même fenêtre de temps.

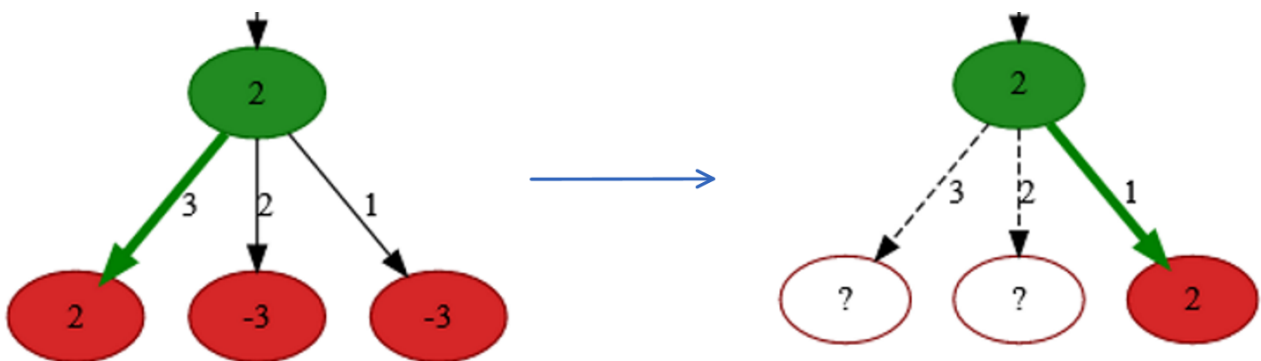


Figure 7: Ordre des coups

Dans un jeu comme les échecs pour avoir le meilleur ordre de coups possibles, il faudrait vérifier d'abord les coups où des pièces sont capturées. Si on se retrouve dans une situation où l'on n'arrive pas à avoir l'ordre des coups optimal, on peut aussi le choisir au hasard ce qui résulte dans une complexité en temps d'environ $O(b^{3*m/4})$.

Encore un facteur très important qui contribue indirectement à la qualité de notre choix de transition, est la fonction d'évaluation. Si pour un état donné elle fait une mauvaise approximation du score, alors le choix de mouvement retourné par l'algorithme minimax ne sera pas optimal. C'est un facteur externe à notre algorithme minimax, mais important à garder en tête.

Implémentation

Le but du projet est d'appliquer l'algorithme minimax avec élagage pour pouvoir visualiser les arbres d'exploration et faire une analyse statistique des résultats. Du coup, il a fallu faire un choix sur un jeu. Mon premier choix était le jeu des échecs, mais la complexité en calcul était trop importante, j'ai fini par choisir le jeu de dames.

Un des objectifs était aussi de se familiariser avec la librairie graphviz qui est très utilisée dans le domaine de visualisation de grandes données. Pour cela, j'ai décidé de faire l'implémentation en JavaScript. Pour manipuler les objets DOM de HTML, j'ai utilisé d3.js (1). Ce dernier possède un module d3-graphviz (2). Pour le jeu de dames j'ai utilisé la librairie draughts.js (3) (jeu de dames en 10x10).

- (1) d3.js est une librairie pour la manipulation de documents basée sur des données
- (2) d3-graphviz.js est un module de d3.js qui crée des graphes décrits dans le langage DOT, en utilisant un port de Graphviz
- (3) draughts.js utilisé pour créer le jeu de dames, jouer des coups, et connaître les transitions possibles depuis un état de jeu donné

Dans les sections précédentes, on a défini et vu l'importance de la fonction d'évaluation, dans mon projet j'ai aussi créé une fonction d'évaluation. Elle assigne une valeur à chaque jeton $+1/-1$ à un jeton blanc/noir respectivement, $+3/-3$ aux dames (jetons doubles) blanc/noir respectivement, puis elle additionne les valeurs de tous les jetons en jeu. Naturellement, si par exemple, le joueur B (jetons noirs) possède plus de jetons que le joueur A, alors le résultat sera négatif.

Le programme accomplit tous les buts du projet, le plus grand problème, c'est que pour accomplir ces mêmes buts, il ne peut pas vraiment être optimal. En effet, on a vu que l'algorithme minimax avait la complexité en temps $O(b^m)$ et en mémoire $O(b * m)$, à l'aide de l'élagage alpha-bêta on a pu réduire la complexité en temps. On peut aussi la réduire en mémoire, si on crée notre arbre d'exploration en même temps qu'on applique l'algorithme minimax. C'est à dire, qu'on peut créer un noeud en mémoire seulement si l'algorithme minimax avec élagage alpha-bêta le visite, ce qui veut dire que les états ignorés, ne sont tout simplement pas créés.

Malheureusement on ne peut pas entièrement en prendre avantage car on veut avoir tout l'arbre développé pour notre analyse statistique, ainsi que pour l'outil de visualisation.

Lors du calcul d'un coup à jouer, les étapes sont les suivantes :

- (1) création de l'arbre d'exploration
- (2) calcul coup optimal à l'aide de l'algorithme minimax: élagage alpha-bêta
- (3) calcul du graphique

Dans le but d'avoir un temps de calcul optimal, on pourrait combler les 3 étapes en une unique étape.

Présentation du logiciel

On commence par l'interface, qui permet à l'utilisateur d'interagir avec le logiciel:



Figure 8: Interface

- Relancer : permet de faire un refresh de la page.
- Faire un coup -> (couleur): le coup optimal basé sur minimax de profondeur donné est calculé et joué, la couleur indique quel joueur fait le coup.
- 10 * Random : permet de faire 10 coups aléatoires.
- Costum Tree : permet de input un arbre souhaité à l'algorithme minimax (Figure 11)
- Deux menus descendants qui permettent de changer la profondeur de l'exploration, chaque joueur en a 1 qui est indépendant de celui de l'autre joueur, ils peuvent être modifiés à tout moment dans la partie.
- En rouge, on voit un trackeur de score.

Le tableau de jeu est un carré divisé en 100 petits carrés de couleur alternée. Affiché dans la première moitié de l'écran:

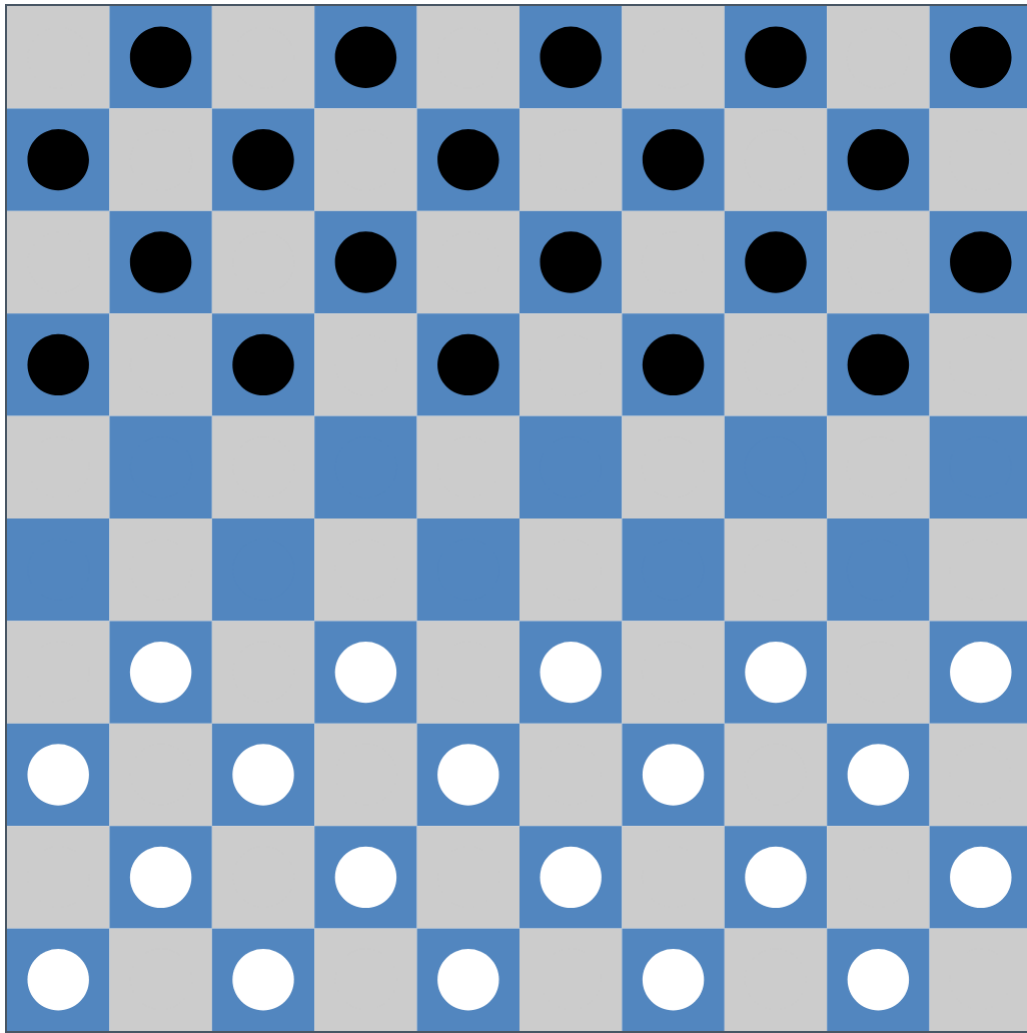


Figure 9: Jeu de dames - Point de départ

Les règles du jeu sont les suivantes:

- Victoire à un joueur quand l'autre joueur n'a plus de jetons.
- Possibilité de créer des dames en avançant jusqu'au côté de l'ennemi.
- Obligation de capturer, en avant et en arrière.

Les arbres d'exploration sont affichés dans l'autre moitié de l'écran:

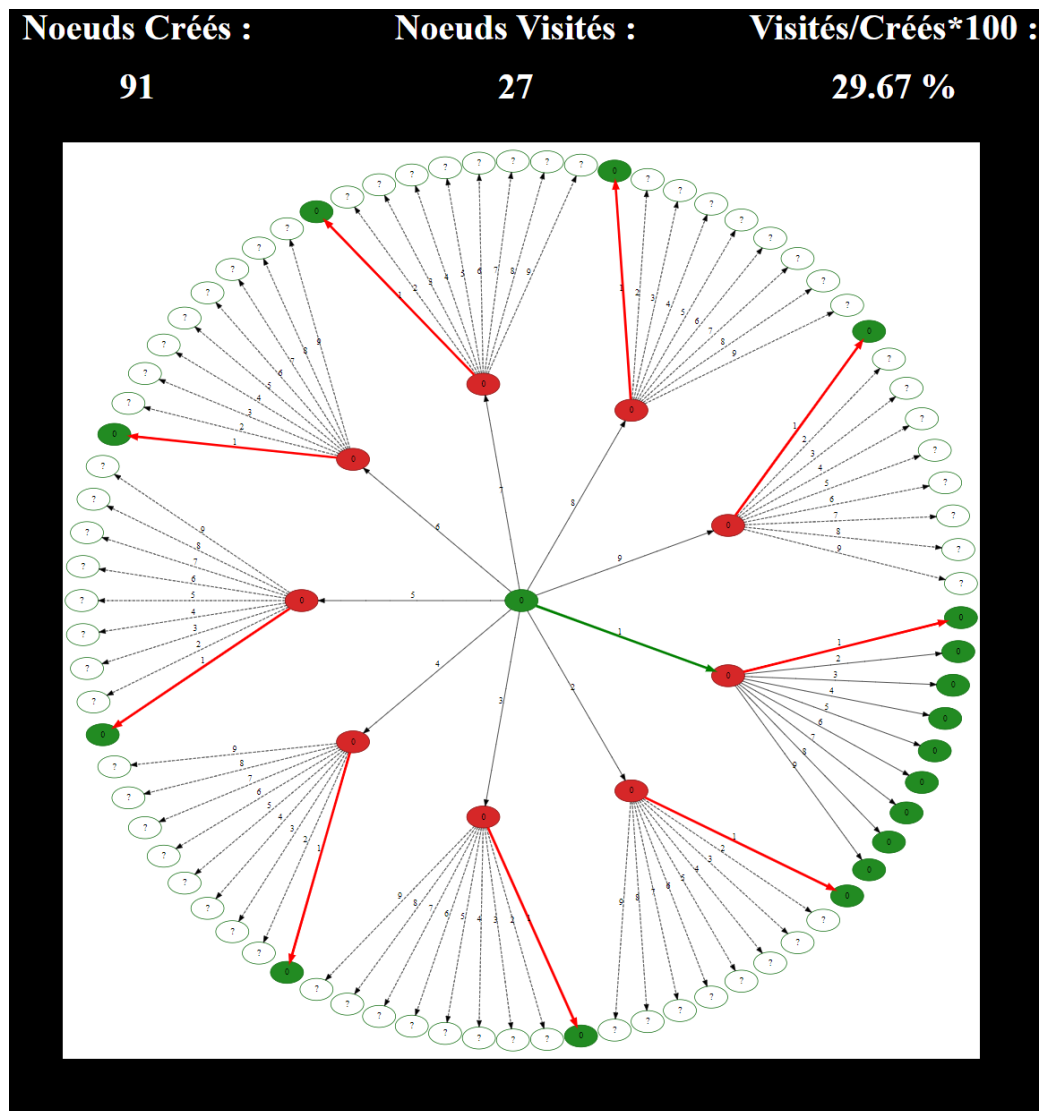


Figure 10: Arbre minimax profondeur 3

Voici la légende du graphe:

- Couleur du noeud symbolise à qui correspond le coup, vert pour max (joueur pièces blanches), rouge pour min (joueur pièces noires).
- Si le noeud est rempli, alors l'algorithme minimax l'a visité, s'il est vide l'algorithme a pu l'élaguer.
- Le chiffre dans le noeud correspond à sa valeur (pour les états finaux), ou alors à la valeur maximale ou minimale (selon le point de vue) qu'on peut atteindre depuis ce noeud (pour les états non finaux).
- Les flèches entre les noeuds symbolisent l'existence d'une transition (un coup) entre les deux états.
- Le chiffre sur les flèches symbolise l'ordre dans lequel l'algorithme a exploré chaque branche.
- Une flèche verte indique que c'est la transition choisie par le joueur max, une flèche rouge indique que c'est la transition choisie par le joueur min, une flèche noire indique que la transition n'a pas été choisie.
- À partir d'une profondeur d'exploration de 4, seul les noeuds visités sont affichés, en raison

Lors qu'un coup est joué par l'algorithme minimax, on a les changements suivants qui sont effectués dans le tableau de jeu:

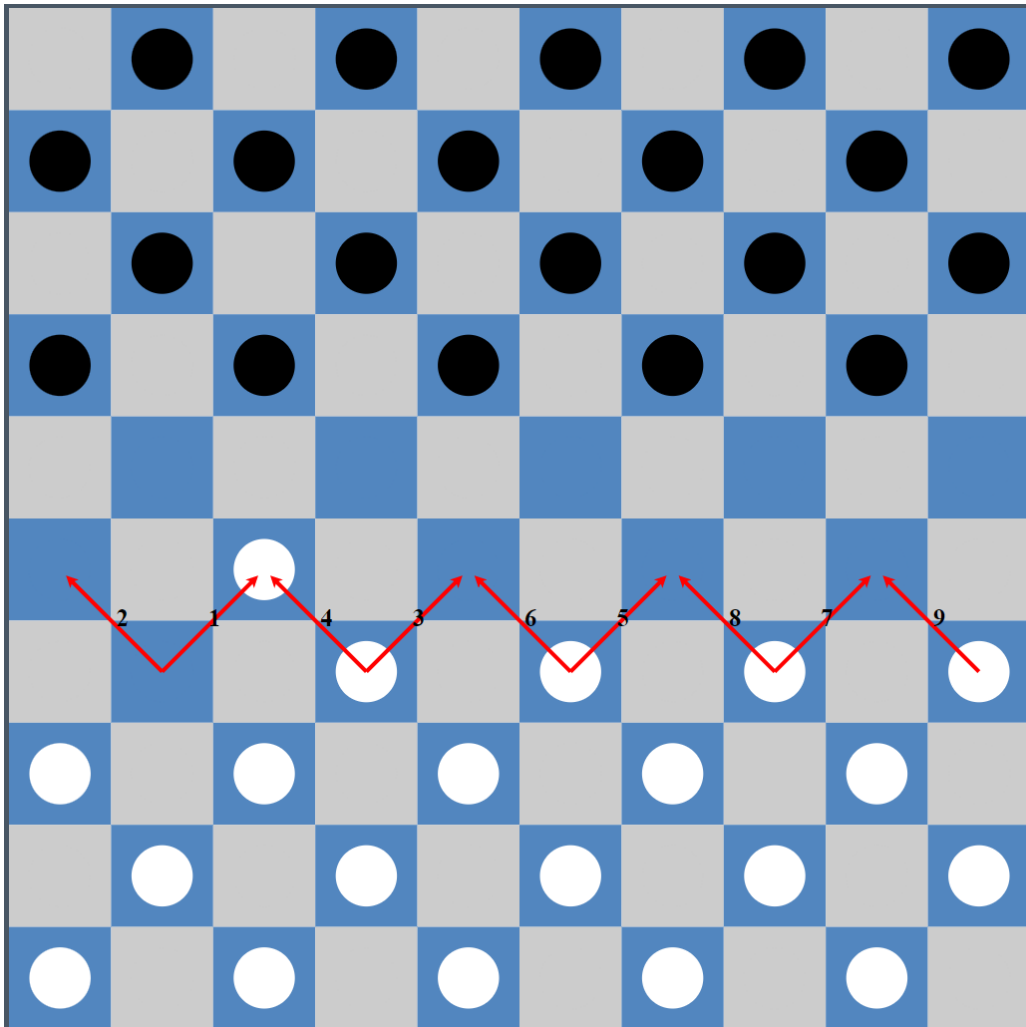


Figure 11: Input d'arbres personnalisés

- Le nouveau coup est joué et affiché
- Des flèches rouges indiquent les coups qui étaient possibles, et les chiffres indiquent à quel transition chaque flèche correspond. Ce sont les mêmes chiffres qui sont visibles dans le graphique (Figure 10).

Costum Tree transforme le tableau des dames dans l'écran suivant:

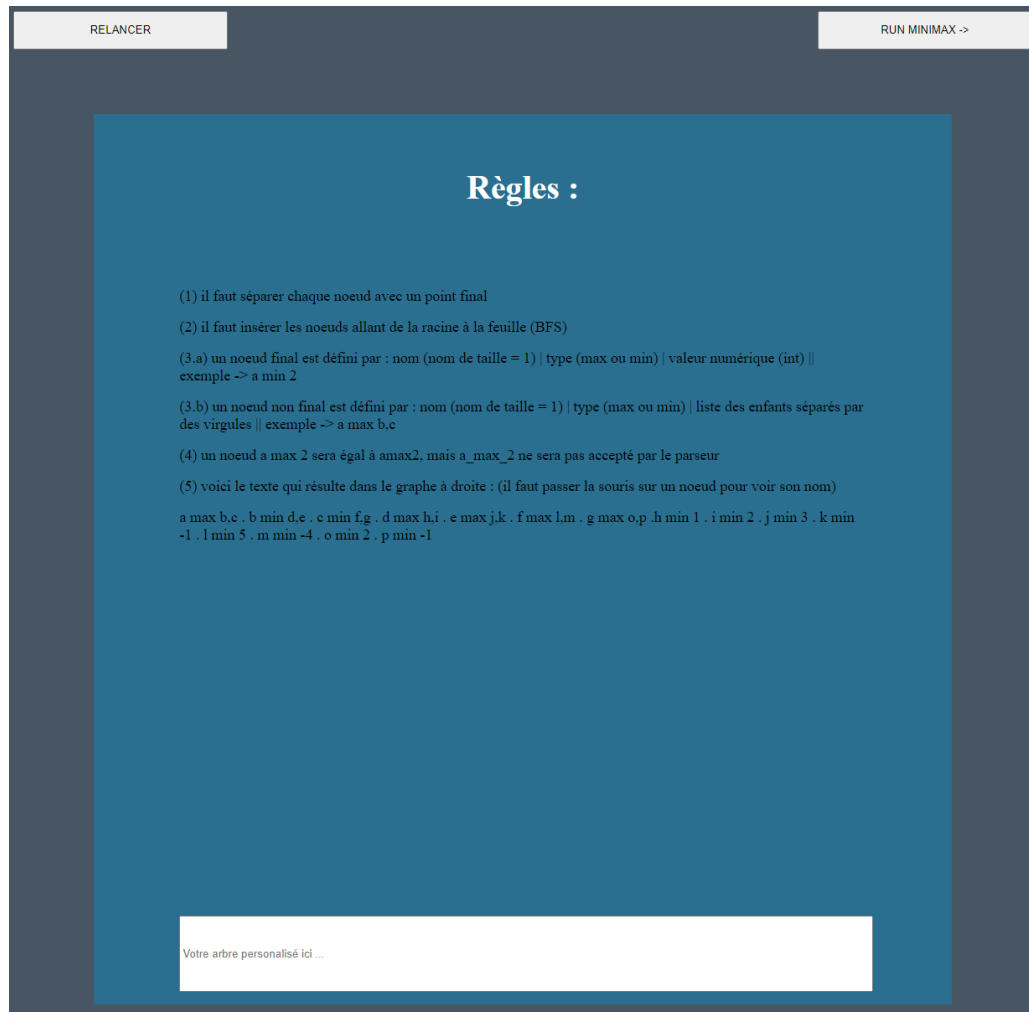


Figure 12: Input d'arbres personnalisés

- Les règles qu'on voit sont en effet un mode d'emploi.
- C'est un simple parser qu'on doit respecter et qui ne donne pas de message d'erreur en cas de mauvais input.
- Relancer : Nous rafraîchi la page entière.
- Run Minimax -> : Fait afficher l'arbre donné comme input dans la fenêtre à droite.

Résultats

Premièrement on va vérifier les avantages de l'élagage alpha-bêta en appliquant l'algorithme avec une profondeur de recherche différente depuis le même état X.

Coups en avance	Noeuds Créés	Noeuds Visités	Noeuds Visités / Noeuds Créés * 100
2	122	34	27.86%
3	1159	175	15.09%
4	11682	235	2.01%
5	111464	715	0.64%
6	513466	2643	0.51%
7	3203600	12178	0.38%

Avec ce tableau on peut créer un graphique, j'ai rajouté au graphique la fonction $\exp(x)$ pour comparer.

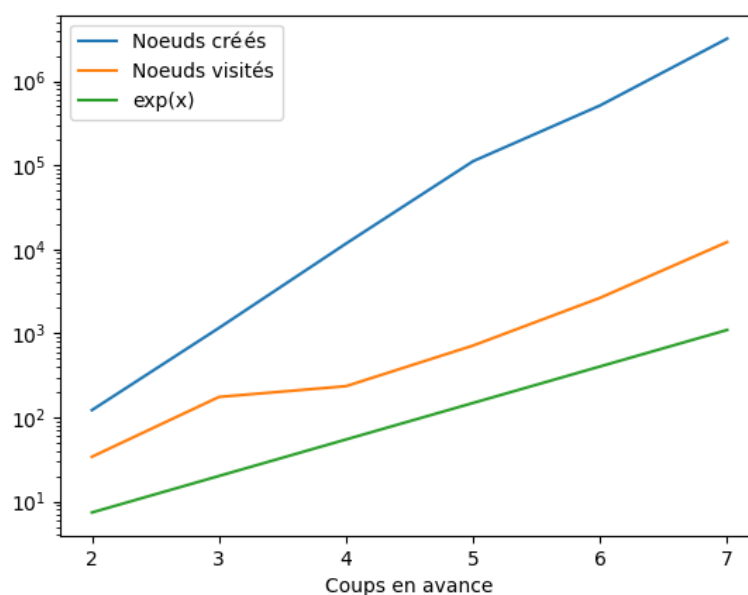


Figure 13: Courbes de croissance noeuds créés et visités

On remarque bien que la courbe des noeuds Créés croît plus que la courbe de noeuds visités. Ce qui vérifie bien notre hypothèse, qui était que l'algorithme minimax avec élagage améliorerait la complexité de l'algorithme minimax simple.

On a vu l'importance de l'ordre dans lequel l'algorithme visite les noeuds. Dans cette section, on va essayer de le démontrer. Vu que la librairie du jeu de dames n'est pas codée par moi-même, je ne peux pas savoir quel est l'ordre dans lequel les transitions sont retournées, si c'est aléatoire, ou s'il y a un ordre spécifique.

Par contre, on peut analyser les résultats avec l'ordre donné par la librairie, puis refaire exactement les mêmes tests en changeant l'ordre des transitions. En théorie, si l'ordre par défaut n'est pas l'optimal, alors on devrait trouver des meilleurs résultats avec les autres ordres. S'il est optimal, on devrait trouver pire avec les autres.

Pour le faire, je vais prendre 5 états aléatoires et lancer à ce moment là l'algorithme minimax, puis analyser les résultats. Tous les tests seront faits avec une profondeur de 5, autrement dit, on voit 4 coups en avance.

Depuis un état aléatoire X, on va lancer l'algorithme minimax avec les 3 ordres différents. Le 1er et 2ème test nous permettent de voir si l'ordre par défaut est optimal, car si il l'est, alors son opposé sera le pire cas possible, c'est à dire le cas où on explore beaucoup plus de noeuds que ce qui est vraiment nécessaire. Finalement, le 3ème test sera avec un ordre de coups aléatoire. Il sert à faire le lien avec la complexité qu'on a vu dans la section théorique, car on sait que l'ordre aléatoire résulte dans une complexité en temps de $O(b^{3*m/4})$.

On va le faire pour 5 états X différents, et voir si les résultats sont consistants à travers les 5 états de départ différents. (voir Figure 13).

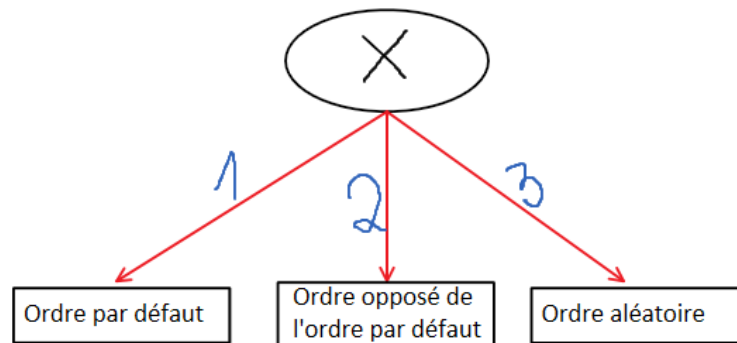


Figure 14: Types de test

Teste (1) : ordre par défaut

État numéro:	Noeuds Créés	Noeuds Visités	Noeuds Visités / Noeuds Créés * 100
1	11682	235	2.01%
2	11388	1202	10.55%
3	15820	1772	11.20%
4	7122	294	4.12%
5	25495	421	1.65%

Teste (2) : ordre opposé de l'ordre par défaut

État numéro:	Noeuds Créés	Noeuds Visités	Noeuds Visités / Noeuds Créés * 100
1	11682	456 (+221)	3.90% (+1.89%)
2	11388	559 (-643)	4.90% (-5.65%)
3	15820	691 (-1081)	4.36% (-6.84%)
4	7122	400 (+106)	5.61% (+1.49%)
5	25495	596 (+175)	2.33% (+0.68%)

Teste (3) : ordre aléatoire

État numéro:	Noeuds Créés	Noeuds Visités	Noeuds Visités / Noeuds Créés * 100
1	11682	372 (+137)	3.18% (+1.17%)
2	11388	576 (-626)	5.05% (-5.50%)
3	15820	561 (-1211)	3.54% (-7.66%)
4	7122	390 (+96)	5.47% (+1.35%)
5	25495	592 (+171)	2.32% (+0.67%)

L'importance de l'ordre des coups devient évidente avec ces résultats. On remarque aussi que l'ordre de coups retourné par la librairie que j'utilise n'est pas la meilleure.

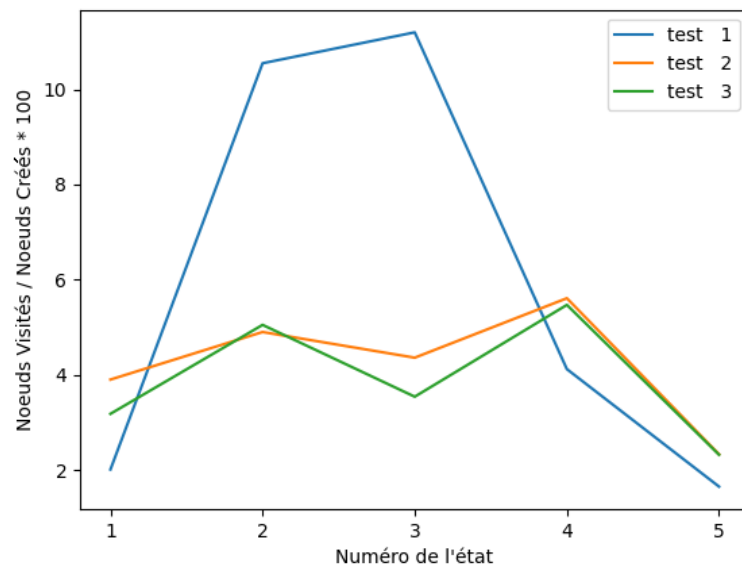


Figure 15: Variance (Noeuds Visités / Noeuds Créés * 100) selon les types de teste

Il faudrait bien sûr faire plus de tests à des différents moments dans le jeu, mais le meilleur ordre a été celui du test 3. C'est celui qui a le moins de variance, il est donc plus prévisible que l'ordre par défaut dans la librairie.

Conclusion

L'algorithme minimax est un algorithme assez simple et logique, qui devient plus compliqué à comprendre une fois qu'on essaye de l'optimiser. Personnellement, j'ai eu de la peine à comprendre les avantages de l'élagage alpha-bêta. Quand je l'ai vu la première fois la théorie était claire mais en pratique, on ne pouvait que ignorer 2-3 noeuds dans les petits graphiques qu'on avait comme exemple, ce qui est bien logique. Il me manquait la possibilité de voir et explorer un grand graphique et voir les avantages. C'était ça le but de ce projet, et il a été accompli, on peut voir deux intelligences artificielles jouer l'une contre l'autre à des degrés d'exploration différents, et comparer tout le long leurs graphiques.

Grâce à ce petit jeu on remarque aussi que l'algorithme marche, et qu'aller le plus profond possible rend l'IA très 'perspicace', car si on laisse les deux IA jouer à des profondeurs différentes, celle qui regarde le plus loin, gagne tout le temps ce qui est attendu, mais intéressant à voir avec ses propres yeux. J'ai trouvé assez amusant de les laisser jouer, et voir leur 'stratégie' mathématique, les deux joueurs ont la notion de jouer avec les jetons proches les uns des autres pour bloquer la possibilité de capture, ainsi que l'idée de sacrifier des pièces pour empêcher la création de dame par l'ennemi.

Malheureusement, il y a des désavantages à l'algorithme minimax avec élagage alpha-bêta très impactants, sa complexité en temps reste très importante (Figure 13). En effet, même si on avait l'ordre de coups le plus optimal possible, on aurait toujours une croissance exponentielle. De plus, l'algorithme dépend de facteurs externes pour son efficacité, pour être capable d'ignorer le plus de noeuds possibles. Par exemple, comme l'ordre de coups (Figure 15), ou encore la qualité de la fonction d'évaluation. Tous ses choix sont bien sûr basés sur la valeur que cette même fonction d'évaluation attribue aux états finaux de l'exploration.

La bonne nouvelle est qu'il y a la possibilité de continuer à améliorer l'algorithme:

- (1) La première chose à améliorer serait la fonction d'évaluation. À l'actualité elle est basée sur la somme des points attribués à chaque pièce en jeu. Mais il est très probable que la façon optimale de jouer soit une autre. Comme par exemple de jouer avec ses jetons ensemble, dans le but de bloquer des éventuelles opportunités d'être capturé par l'ennemi. Il faudrait attribuer un score plus élevé à une configuration de jeu qui respecterait cette tactique.
- (2) Il existe la variante du minimax avec élagage, la variante 'cut-off', qui consiste à arrêter l'exploration de certaines branches avant d'atteindre la profondeur maximale, ce 'cut-off' est basé selon des critères précis. En l'utilisant, on ignorerait encore plus de noeuds que dans la variante élagage alpha-bêta. Par contre, on ne pourra pas être sûr d'avoir le coup optimal, car si notre fonction qui décide où 'cut-off' n'est pas bonne, il se peut qu'on n'explore pas les meilleures branches.
- (3) Dans le but de réduire la complexité en mémoire, on peut aussi implémenter la variante élagage-avant, dans cette variante on cherche les transitions qui ont peu de probabilité d'être choisies, l'algorithme ne crée même pas les noeuds auxquels on arriverait avec ces transitions, c'est une sorte de 'cut-off' pour la mémoire plutôt que le temps.

Dans ces deux variantes, on dépend encore plus de la fonction qui décide où 'cut-off' l'exploration. Contrairement à la variante élagage alpha-bêta, elles n'assurent pas que le coup choisi soit le meilleur.

Références

@book title = Intelligence Artificielle, volume = 3e éd., isbn = 978-2-7440-7455-4, pages = 1198, author = Russell Stuart, Peter Norvig, date = 2010,
language = français