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 la 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 que 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 algorithmes minimax à des profondeurs de recherche différentes, pour regarder l'efficacité de l'élagage alpha-bêta j'afficherais 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 coté, 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 la 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 un jeu où un joueur A affronte le joueur B, et 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 quantifiable/déterministe, le morpion les échecs et les dames sont des exemples de jeux auxquels on peut appliquer cet algorithme du minmax. L'idée de l'algorithme est très simple, le joueur A regarde tous les états que l'on peut attendre depuis un état de jeu donné, puis pour tous les états trouvés, il refait la même chose, mais en se mettant à la place du jouer B, une fois qu'on arrivé 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 quel est l'état qui avantage plus ou moins un joueur donné, il faut avoir un moyen de comptabiliser un état de jeu, et savoir s'il est à l'avantage du joueur A ou du joueur B.

Regardons l'exemple suivant simple (Figure 1), la couleur verte symbolise un état où il est au joueur A de jouer, et la couleur rouge un état où il est au joueur B de jouer, et les chiffres dans les états symbolisent le score en ce même état, le score est la quantification de l'état, et les chiffres en bleu le nom de l'état.

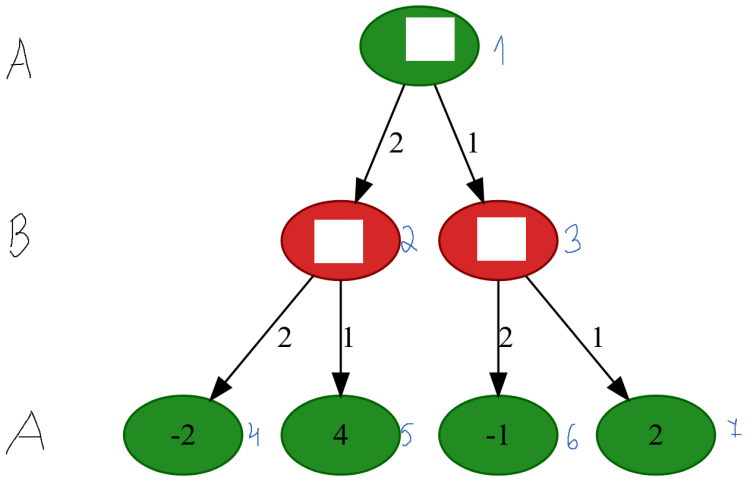

Figure 1: Exemple simple: partie 1

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 appliqué pour l'état 3. Le but du joueur A est de maximiser la valeur final, 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 nos 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 je joueur B se retrouve dans l'état 2, il va joueur la transition 2, ce qui veut dire qu'on peut quantifier le score des états non finaux 2 et 3, en réfléchissant à quel coup B choisirait.

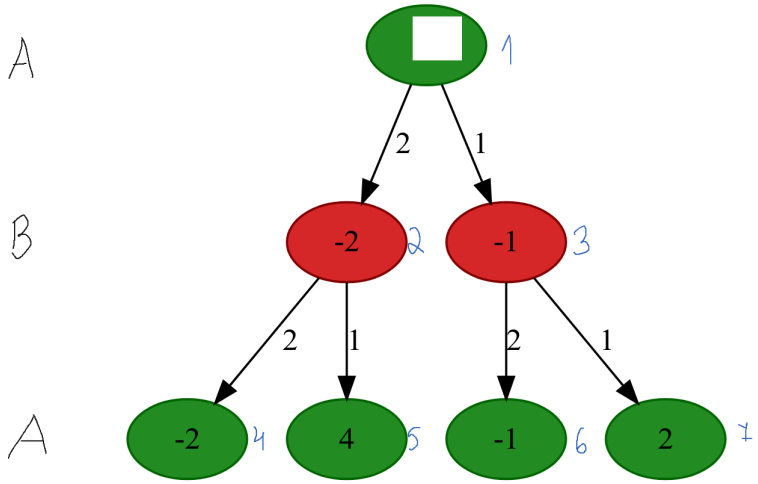

Figure 2: Exemple simple: partie 2

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 refflexion 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. En effet les états finaux ont un score déterministe alors que les états non finaux on une espérance de score qui est réaliste. (Comme supposé, on part du principe que B fera le meilleur coup possible pour lui même).

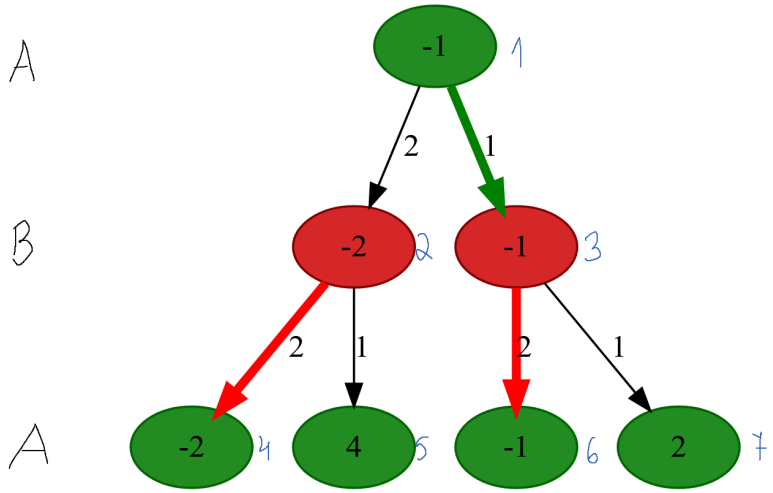

Figure 3: Exemple simple: partie finale

Voici l'arbre d'exploration minimax de cet exemple simple (Figure 3). On comprend aussi le nom de l'algorithme, il explore l'arbre en choisissant le maximum et le minimum en alternance. 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 dés le début.

Cette techniques peut être appliqué 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 une moyenne 100 coups joués (50 par jouer), 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, pour tout état donné x on pourra, en moyenne, atteindre b états différents, et 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 $35^4 = 1'500'625$ d'états différents $< 10^{154}$. Cela représente encore un très grand nombre d'états pour un ordinateur, on va donc essayer d'optimiser l'exploration.

Minimax: Élagage alpha-bêta

L'exploration d'un arbre minimax est exponentielle, on ne pourra pas complètement éliminer 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 ensemble (Figure 4), 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, selon quel joueur fait le choix et la flèche colorié indique d'où le noeud prend sa 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.

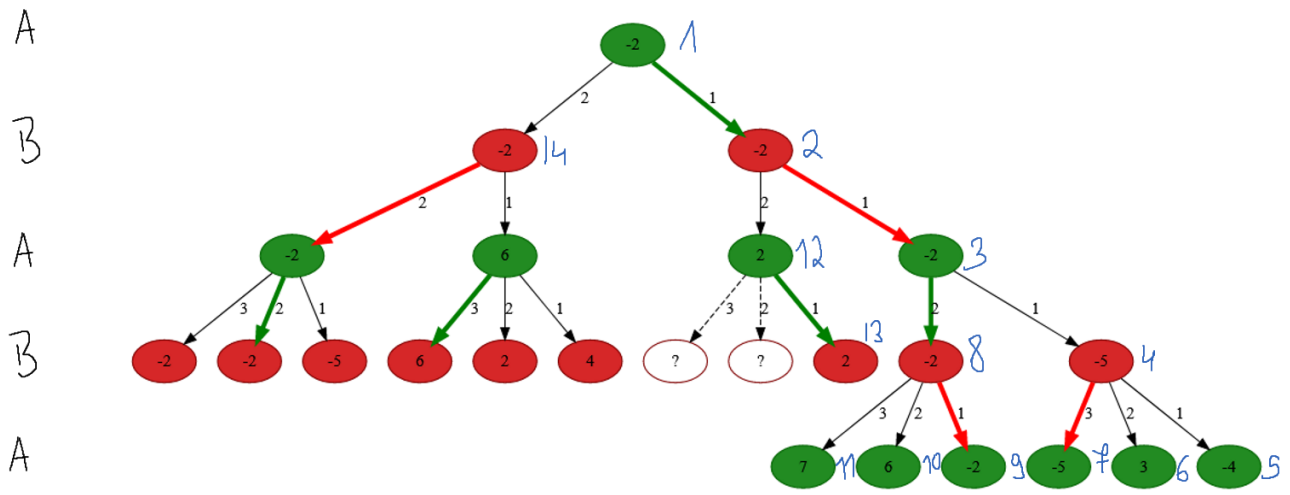

Figure 4: Élagage alpha-bêta

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 lors qu'on arrive au noeud 12, le joueur B étant au noeud 2 à 2 options possibles, la première le mène au noeud 3 qui 'vaut' -2, la deuxième n'est pas encore exploré, mais même sans l'explorer a déjà la garanti que s'il prend la première option, le score sera de -2. Puis il explore le noeud 12, et bien sur arrive au noeud final 13 qui vaut 2, le joueur B peut maintenant s'arrêter, en effet, si le joueur B choisit la deuxième option il sait dejà que le joueur A à son tour pourra choisir d'aller dans l'état 13, qui vaut 2, et $2 > -2$, Sachant ça, il n'a pas besoin de visiter les autres enfants du noeud 12, car comme supposé, A est un bon joueur, du coup B sait qui se A se trouve dans l'état 12, il aura la possibilité de joueur un coup qui mènera à un score qui vaut 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, elle est la l'élégance de l'algorithme alpha-bêta, le plus intéressants c'est que si les 2 noeuds ignorés avaient des enfants, on aurait pu les ignorer ici, du coup dans un arbre plus profond, avec un branchement plus important, on pourra, en théorie, ignorer de plus en plus des noeuds.

Regardons encore un exemple (Figure 5), qui nous fera comprendre l'importance de l'ordre.

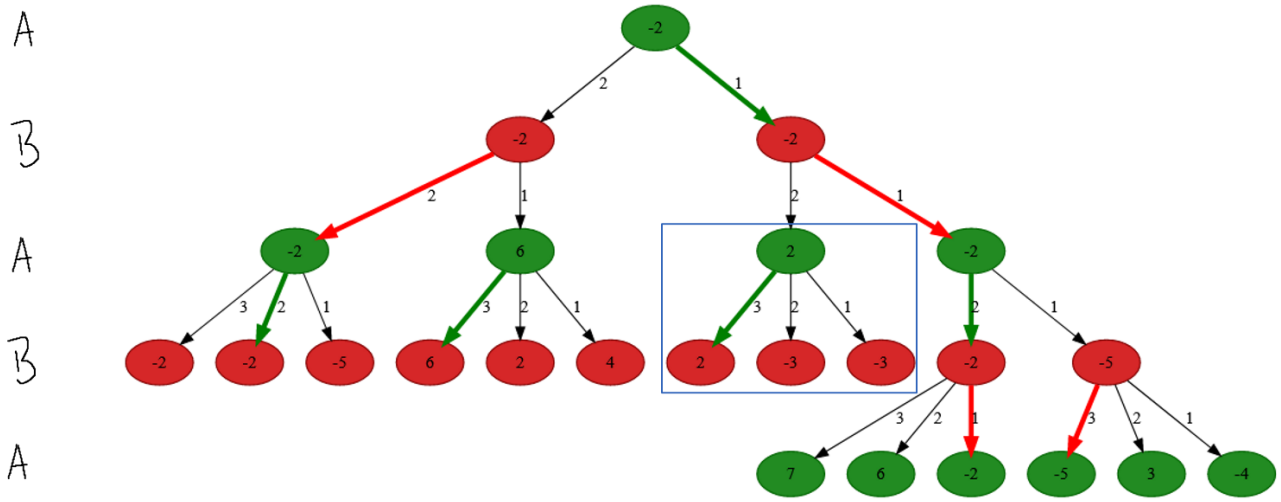

Figure 5: Élagage alpha-bêta 2

En regardant la figure 5, on remarque une ressemblance avec l'exemple précédent, la seule différence c'est que ici on n'a pas pu ignorer des noeuds, mais l'algorithme est le même, et l'arbre est aussi presque le même que 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 amélioré, peut avoir une chance d'ignorer quelques noeuds. Le problème est que ça ne dépend pas de l'algorithme lui même, mais entièrement de l'ordre dans lequel les enfants de chaque noeud sont organisés.

En théorie, disons qu'il est possible d'ordonner les coups de façon à maximiser les noeuds élagués (Figure 6), alors on pourrait améliorer la complexité du minimax basique, 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 parfait 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.

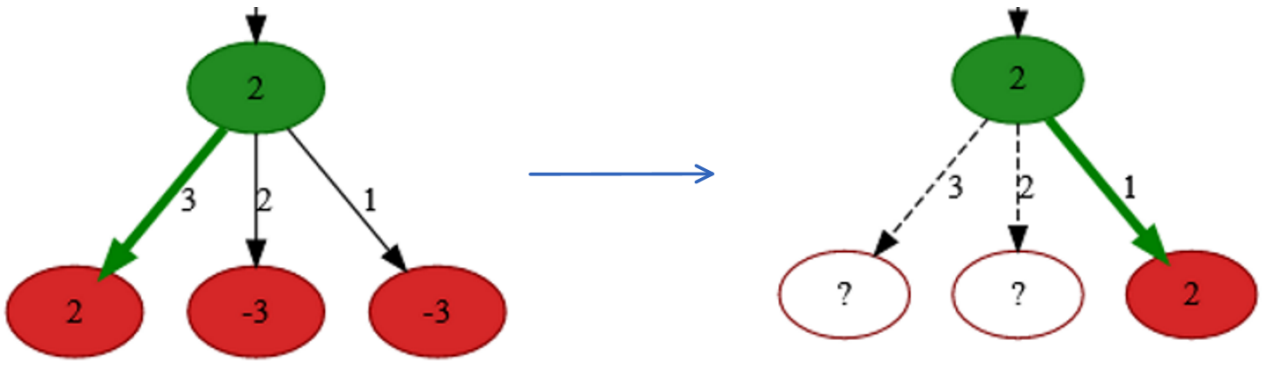

Figure 6: Ordre des coups

Dans un jeu comme les échecs pour avoir le meilleur ordre des coup possible, il faudrait vérifier d'abord les coups où des pièces sont capturés. Si on se retrouve dans une situation où l'on n'arrive pas à avoir l'ordre des coups optimale, 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 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 faut faire un choix sur un jeu. Le premier choix était le jeu des échecs, mais la complexité en calcul était trop importante, j'ai fini par choisir le jeu des dames.

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

(1) d3.js est un librairie pour la manipulation de documents basé sur des données

(2) d3-graphviz.js est un module de d3.js qui crée des graphs décrits dans le langage DOT, en utilisant un port de Graphviz

(3) draughts.js utilisé pour créer le jeu des dames, jouer des coups, et connaître les transitions possibles depuis un état de jeu donné

Présentation du produit

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

Figure 7: Interface

- Relancer : permet de faire un refresh dans la page.

- Faire un coup : le coup optimale basé sur minimax de profondeur donné est calculé et joué.

- 10 * Random : permet de faire 10 coups aléatoires.

- Costum Tree : permet de input une arbre souhaité à l'algorithme minimax (Figure 10)

- 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 socre.

Le tableau de jeu est un carré divisé en 100 petits carrés qui alternent en couleur:

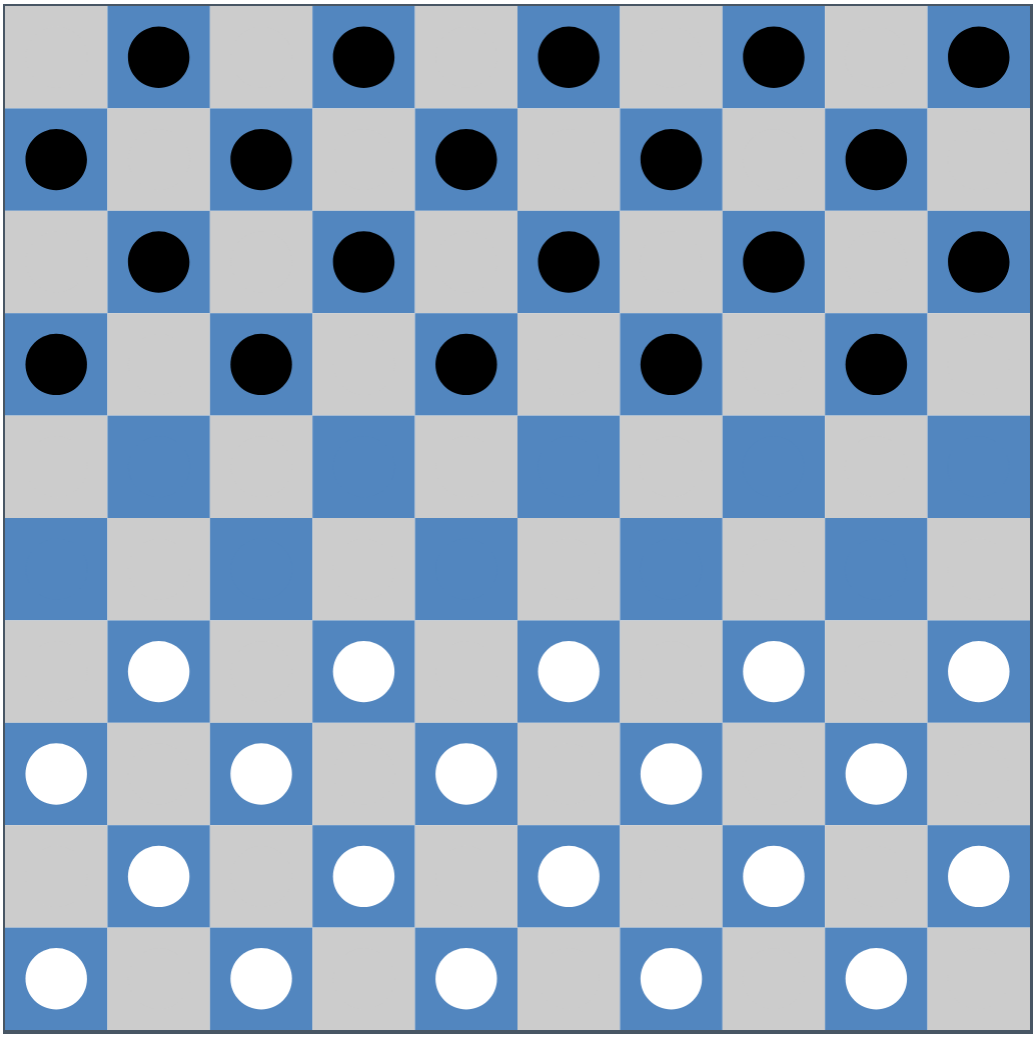

Figure 8: Jeu de dames Départ

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 coté de l'ennemi.
- Obligation de capturer, en avant et en arrière.

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

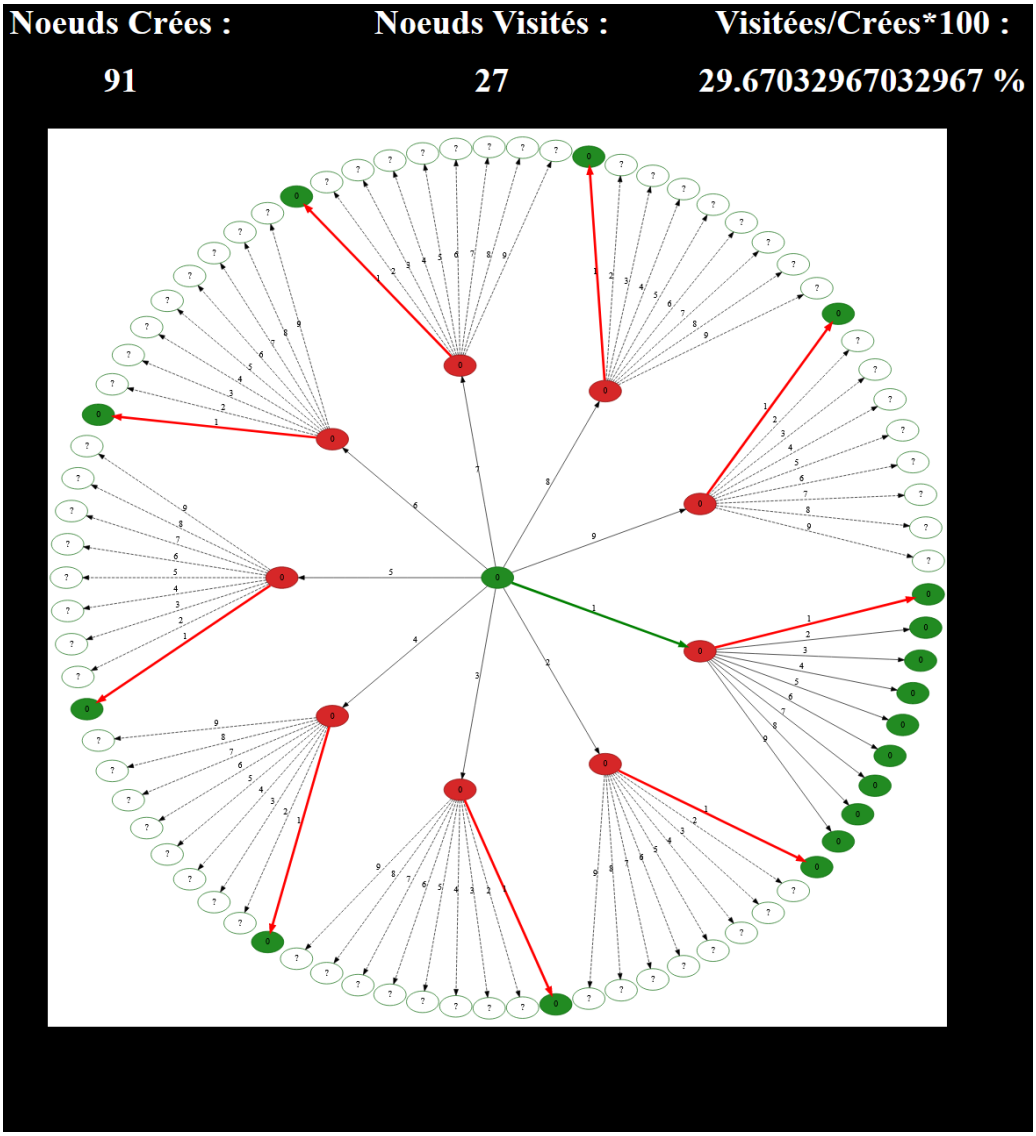

Figure 9: Arbre minimax profondeur 3

Voici la lègende du graph:

- 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 minimal (selon le point de vue) qu'on peut atteindre depuis ce noeud (pour les états non finaux).

- Les flèches entre noeuds symbolisent l'existence d'une transition (un coup) entre les deux états.

- Le chiffre dans 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 noir indique que la transition n'a pas été choisie.

- À partir d'une profondeur d'exploration de 4, les noeuds non visités ne sont plus affichés.

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

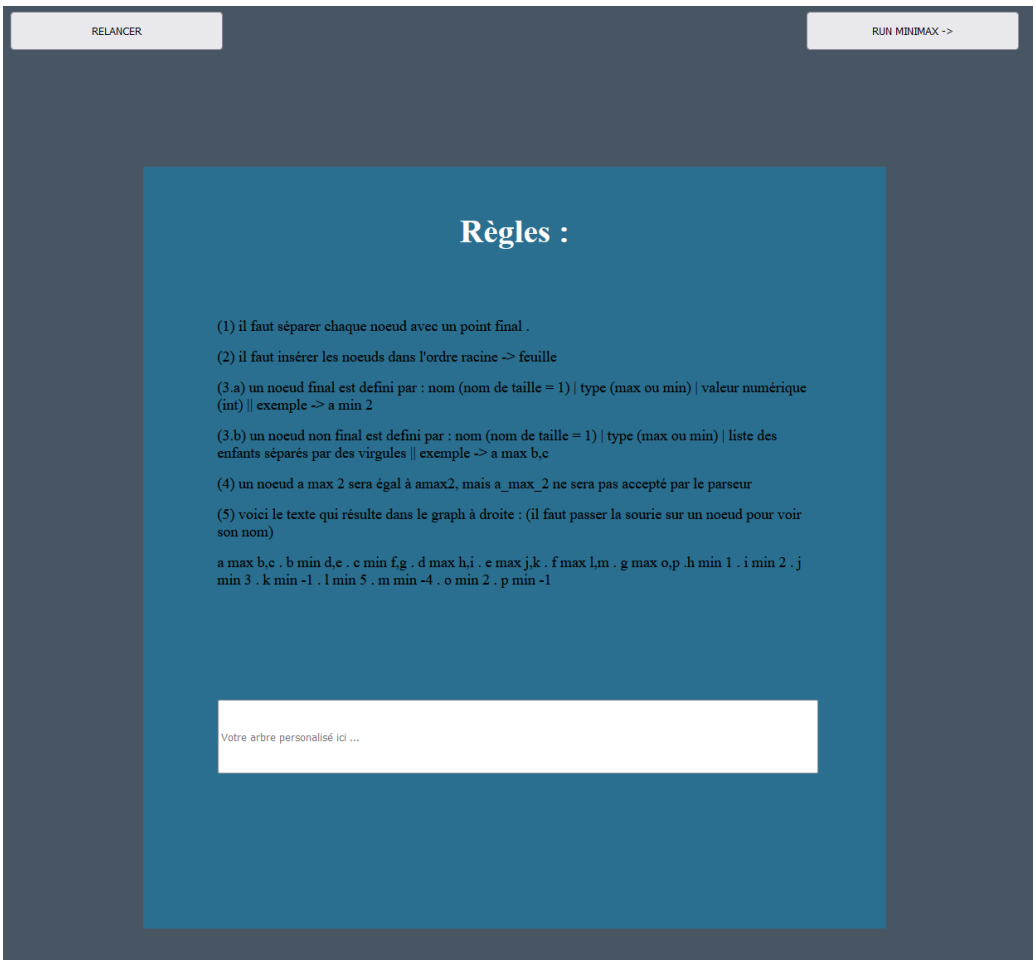

Figure 10: Input d'arbres personnalises

- 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 donnée 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 Crées	Noeuds Visités	Noeuds Visités / Noeuds Crées * 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.

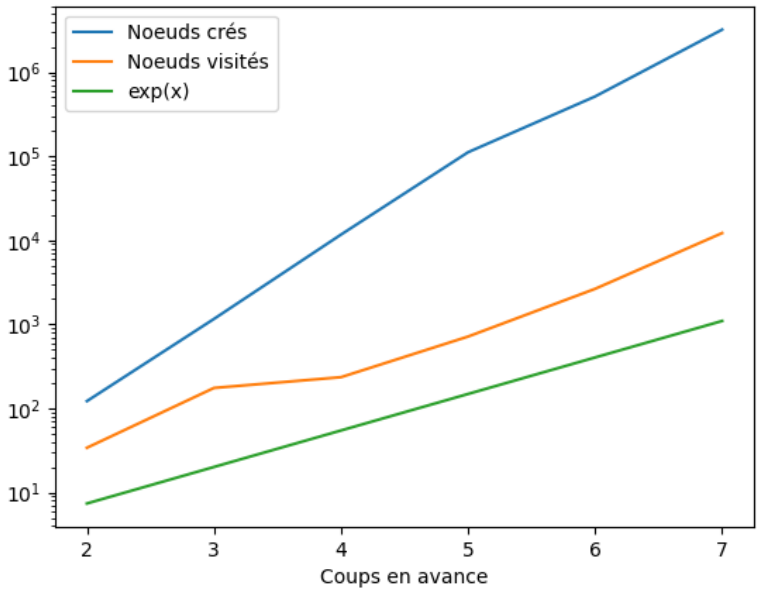

Figure 11: Courbes de croissance noeuds crées et visités

On remarque bien que la courbe des noeuds crées croît plus que le courbe de noeuds visités. Ce qui vérifie bien notre hypothèse.

Nous avons aussi parlé de l'importance de l'ordre dans laquelle l'algorithme visite les noeuds, dans cette sections on va essayer de le démontrer. Vu que la librairie du jeu des dames n'est pas codé 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 testes en changeant l'ordre des transitions.

Pour le faire je vais prendre 10 états aléatoires, et lancer à ce moment la l'algorithme minimax, puis analyser les résultats. Tous les testes seront faits avec une profondeur de 5, autrement dit, on voit 4 coups en avance.

En théorie, si l'ordre par défaut n'est pas l'optimale, alors on devrait trouver des meilleurs résultats avec les autres ordres, s'il est optimale, on devrait trouver pire avec les autres.

Depuis un état aléatoire X, on va lancer l'algorithme minimax avec les 3 ordres différents. 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 12). Le 1er et 2ème teste nous permettent de voir si l'ordre par défaut est optimale, car si elle l'est, alors son opposé sera très le pire cas possible, le cas où on explore beaucoup plus de noeuds que ce qu'il

est vraiment nécessaire. Finalement le teste d'ordre aléatoire sert à faire le lien avec la complexité qu'on a vu dans la section théorique, car on sait que l'ordre aléatoire résultat dans une complexité en temps de $O(b^{3*m/4})$.

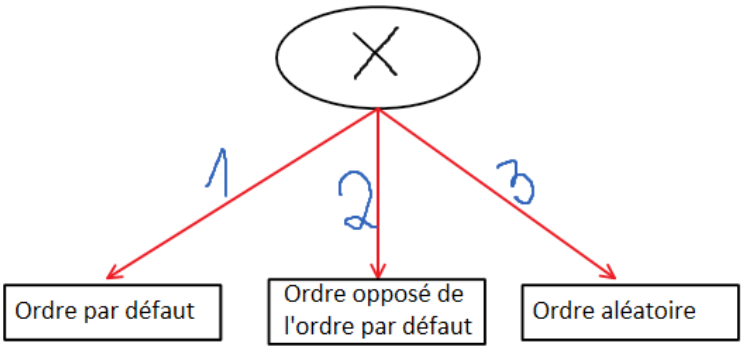

Figure 12: Types de teste

Teste (1) : ordre par défaut

État numéro:	Noeuds Crées	Noeuds Visités	Noeuds Visités / Noeuds Crées * 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 Crées	Noeuds Visités	Noeuds Visités / Noeuds Crées * 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 Crées	Noeuds Visités	Noeuds Visités / Noeuds Crées * 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.

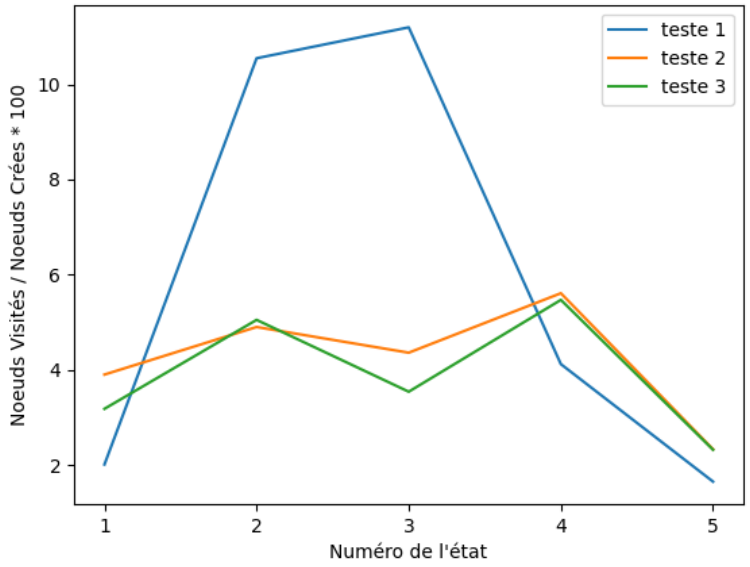

Figure 13: Variance (Noeuds Visités / Noeuds Crées * 100) selon les types de teste

Il faudrait bien sur faire plus de testes à des différents moments dans le jeu, mais le meilleur ordre a été celui du teste 3, c'est celui qui a le moins de variance, il est donc plus prévisible que l'ordre par défaut dans la librairie. L'ordre du teste 3 est celui qui correspond à une complexité théorique de $O(b^{3*m/4})$.

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 clair, mais en pratique, on 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 accomplie, 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 sympa 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 un 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 11), même si on avait l'ordre de coups le plus optimale possible, on aurait toujours une croissance exponentielle. De plus, l'algorithme dépend de facteurs externes pour son efficacité, comme l'ordre des coups (Figure 13) pour être capable d'ignorer le plus de noeuds possibles, ou encore la qualité de la fonction d'évaluation car l'IA n'a en effet aucune intelligence, tout ses coups sont bien sur basé 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 place pour continuer à améliorer l'algorithme:

(1) 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. On ignorerait encore plus de noeuds que dans la variante élagage alpha-bêta, par contre on ne pourra pas être sur d'avoir le coup optimale, car si notre fonction qui décide où 'cut-off' n'est pas bonne, il se peut qu'on n'explore pas les meilleurs branches.

(2) 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 crée même pas les noeuds auxquels on arriverait avec ces transitions, c'est un 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 choisie soit le meilleur.