

Rapport du Projet

**Artificial Intelligence**

Préparée par:

Fadila ALAOUI (2-IIN)

Safaa AMIR (3-ICSD)

Ibtissam LABYADY (23-ICSD)

Sommaire

[Qu'est-ce que l'algorithme Minimax ? 4](#_Toc120098821)

[Comment fonctionne l'algorithme ? 5](#_Toc120098822)

[Psuedocode: 8](#_Toc120098823)

[Optimisation 8](#_Toc120098824)

[Élagage alpha-bêta 9](#_Toc120098825)

[Conclusion 11](#_Toc120098826)

[Webographie 13](#_Toc120098827)

# Introduction

Depuis l'avènement de l'intelligence artificielle (IA), le jeu est l'une des applications les plus intéressantes de l'IA. Les premiers programmes d'échecs ont été écrits par Claude Shannon et par Alan Turing en 1950, presque aussitôt que les ordinateurs sont devenus programmables. Des jeux comme les échecs, le tic-tac-toe et le go sont intéressants car ils offrent une pure abstraction de la compétition entre les deux armées. C'est cette abstraction qui fait du jeu un domaine attractif pour la recherche en IA

Tic-Tac-Toe (ainsi que beaucoup d'autres jeux) implique de regarder devant et d'essayer de comprendre ce que la personne qui joue contre vous pourrait faire ensuite.

RÈGLES DU TIC-TAC-TOE

1. Le jeu se joue sur une grille de 3 cases sur 3 cases.

2. Vous êtes X, votre ami (ou l'ordinateur dans ce cas) est O. Les joueurs placent à tour de rôle leurs marques dans des cases vides.

3. Le premier joueur à obtenir 3 de ses marques d'affilée (en haut, en bas, en travers ou en diagonale) est le gagnant.

4. Lorsque les 9 cases sont pleines, le jeu est terminé. Si aucun joueur n'a 3 points d'affilée, la partie se termine par une égalité.

En haut à gauche En haut au centre En haut à droite

Milieu-gauche Milieu-centre Milieu-droit

En bas à gauche En bas au centre En bas à droite

# 2-Qu'est-ce que l'algorithme Minimax ?

Minimax est un algorithme récursif qui est utilisé pour choisir un coup optimal pour un joueur en supposant que l'autre joueur joue également de manière optimale. Il est utilisé dans des jeux tels que le tic-tac-toe, le go, les échecs, l'Isola, les dames et de nombreux autres jeux à deux joueurs. De tels jeux sont appelés jeux d'informations parfaits car il est possible de voir tous les coups possibles d'un jeu particulier. Il peut y avoir des jeux à deux joueurs qui ne sont pas d'informations parfaites comme le Scrabble parce que le mouvement de l'adversaire ne peut pas être prédit. C'est similaire à la façon dont nous pensons lorsque nous jouons à un jeu : "si je fais ce mouvement, alors mon adversaire ne peut faire que ces mouvements", et ainsi de suite. Minimax est appelé ainsi car il aide à minimiser la perte lorsque l'autre joueur choisit la stratégie ayant la perte maximale.

Terminologie

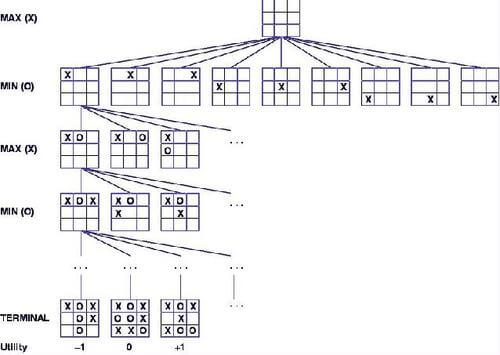
* **Game Tree** : C'est une structure sous forme d'arbre constituée de tous les coups possibles qui permet de passer d'un état du jeu à l'état suivant.

Un jeu peut être défini comme un problème de recherche avec les composants suivants :

* **État initial** : Il comprend la position du plateau et indique à qui appartient le coup.
* **Fonction successeur** : elle définit les coups autorisés qu'un joueur peut effectuer.
* **Etat terminal** : C'est la position du plateau à la fin de la partie.
* **Fonction d'utilité** : C'est une fonction qui attribue une valeur numérique au résultat d'un jeu. Par exemple, aux échecs ou au tic-tac-toe, le résultat est soit une victoire, soit une défaite, soit un match nul, et ceux-ci peuvent être représentés par les valeurs +1, -1 ou 0, respectivement. Il existe des jeux qui ont une gamme beaucoup plus large de résultats possibles ; par exemple, les utilités au backgammon variant de +192 à -192. Une fonction d'utilité peut également être appelée fonction de gain.

# 3-Comment fonctionne l'algorithme ?

Il y a deux joueurs impliqués dans un jeu, appelés MIN et MAX. Le joueur MAX essaie d'obtenir le score le plus élevé possible et MIN essaie d'obtenir le score le plus bas possible, c'est-à-dire que MIN et MAX essaient d'agir à l'opposé l'un de l 'autre. Le processus général de l'algorithme Minimax est le suivant :

**Étape 1** : Tout d'abord, générez l'arborescence complète du jeu en commençant par la position actuelle du jeu jusqu'aux états terminaux. Voici à quoi ressemble l'arbre de jeu pour le jeu tic-tac-toe.[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/tic-tac-toe.jpg)

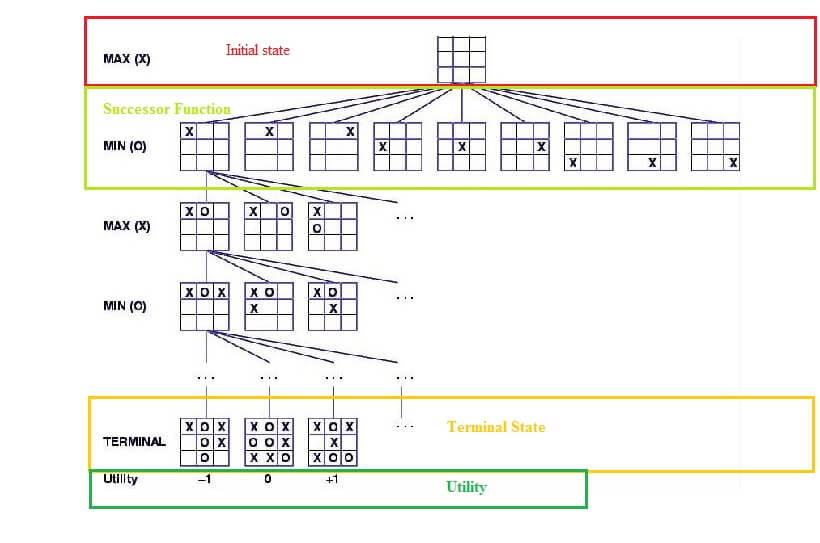
Comprenons la terminologie définie en termes de diagramme ci-dessus.

L’état initial est le premier calque qui définit que le plateau est vierge, c’est au tour de MAX de jouer.

La fonction successeur répertorie tous les mouvements successeurs possibles. Il est défini pour toutes les couches de l’arborescence.

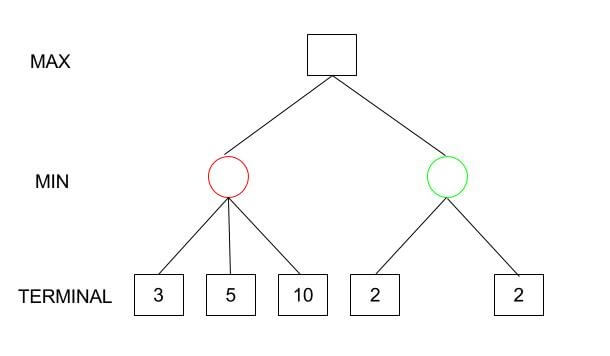
L’état terminal est la dernière couche de l’arbre qui montre l’état final, c’est-à-dire si le joueur MAX gagne, perd ou fait jeu égal avec l’adversaire.

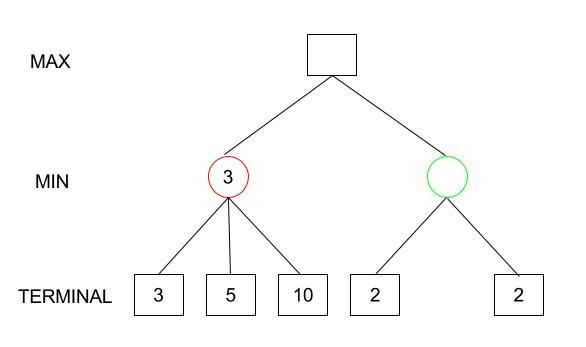
Dans ce cas, les utilitaires pour les états terminaux sont 1, 0 et -1, comme indiqué précédemment, et ils peuvent également être utilisés pour déterminer les utilitaires des autres nœuds.

.[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/ttt-minimax.jpg)

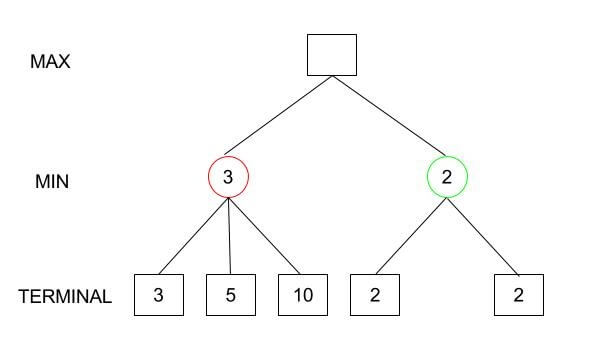
**Étape 2** : Appliquez la fonction d'utilité pour obtenir les valeurs d'utilité pour tous les états du terminal.

**Étape 3** : Déterminer les utilitaires de nœuds supérieurs à l'aide des utilitaires de nœuds connectés. Par exemple, dans le diagramme ci-dessous, nous avons les utilitaires pour les états terminaux écrits dans les carrés..

[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/Minimax-1.jpg)

Calculons l'utilité du nœud gauche (rouge) de la couche au-dessus du terminal. Puisqu'il s'agit du coup du joueur MIN, on choisira le minimum de tous les utilitaires. Dans ce cas, nous devons évaluer MIN{3, 5, 10}, dont nous savons qu'il est certainement égal à 3. L'utilité du nœud rouge est donc de 3.  


De même, pour le nœud vert dans la même couche, nous devrons évaluer MIN{2,2} qui est 2.

[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/Minimax-3.jpg)

**Étape 4**: Calculez les valeurs d’utilité à l’aide de feuilles en considérant une couche à la fois jusqu’à la racine de l’arbre.  
**Étape 5**: Finalement, toutes les valeurs sauvegardées atteignent la racine de l’arbre, c’est-à-dire le point le plus haut. À ce stade, MAX doit choisir la valeur la plus élevée.

Dans notre cas, nous n’avons que 3 couches, nous avons donc immédiatement atteint la racine, mais dans les jeux réels, il y aura beaucoup plus de couches et de nœuds. Nous devons donc évaluer MAX{3,2} qui est 3.

Par conséquent, le meilleur mouvement d’ouverture pour MAX est le nœud gauche (ou le rouge). Ce mouvement est appelé la décision minimax car il maximise l’utilité en supposant que l’adversaire joue également de manière optimale pour le minimiser.

Pour résumer,

Décision Minimax = MAX{MIN{3,5,10},MIN{2,2}}= MAX{3,2}= 3

### Psuedocode:

function minimax(node, depth, maximizingPlayer)

if depth = 0 or node is a terminal node

return the utility of the node

if maximizingPlayer

bestValue := -infinie

for each child of node

v= minimax(child, depth + 1, FALSE)

bestValue := max(bestValue, v)

return bestValue

else (\* minimizing player \*)

bestValue := +infinie

for each child of node

v := minimax(child, depth + 1, TRUE)

bestValue := min(bestValue, v)

return bestValue

**PROPRIÉTÉS DE L'ALGORITHME:**

1- Complet : Il est capable de trouver la solution (si elle existe), à partir de l'arbre de recherche fini.

2-Optimal : Il s'avère être optimal, si les choix des deux joueurs s'avèrent optimisés.

3-Complexité temporelle(time complexity) : Puisqu'il est observé de suivre la recherche en profondeur pour traverser l'arbre de jeu(Depth- First Search), la complexité est O(b^m);"b" est le le nombre d'enfants à chaque nœud, et la profondeur maximale de l'arbre est définie par "m".

4-Complexité spatiale : elle est similaire à celle de la recherche en profondeur(Depth-First Search ), c'est-à-dire O(bm).

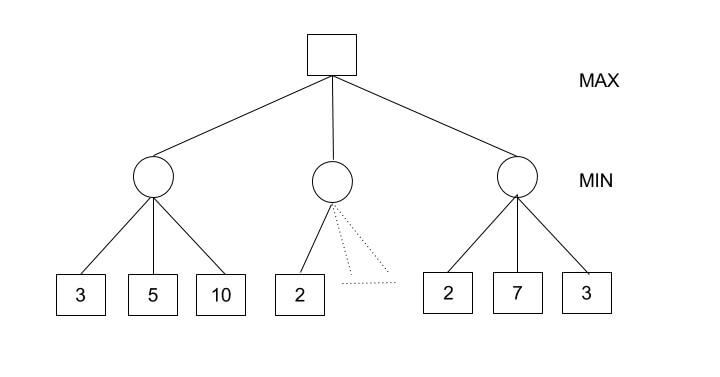
## 4-Élagage alpha-bêta

Les arbres de jeu sont, en général, très longs à construire, et ce n’est que pour les jeux simples qu’ils peuvent être générés en peu de temps.

Pour enrayer cette situation, quelques optimisations peuvent être ajoutées à l’algorithme.

IL est viable de trouver la décision minimax réelle sans même regarder chaque nœud de l’arbre de jeu. Par conséquent, nous éliminons les nœuds de l’arbre sans analyser, et ce processus s’appelle élagage alpha-beta.

Si nous appliquons l’élagage alpha-bêta à un algorithme minimax standard, il renvoie le même mouvement que le standard, mais il supprime (élague) tous les nœuds qui n’affectent peut-être pas la décision finale.

Comprenons d’abord l’intuition derrière cela, puis nous formaliserons l’algorithme. Supposons que nous ayons l’arbre de jeu suivant:  
[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/Minimax-algorithm-with-alpha-beta-pruning-for-AI-1.jpg)

Dans ce cas,

Minimax Decision = MAX{MIN{3,5,10}, MIN{2,a,b}, MIN{2,7,3}}  
 = MAX{3,c,2}  
 = 3

Comment pourrions-nous calculer le maximum avec une valeur manquante?

Voici l’astuce. MIN{2,a,b} serait certainement inférieur ou égal à 2, c’est-à-dire que c=2 et donc MAX{3,c,2} doit être 3.

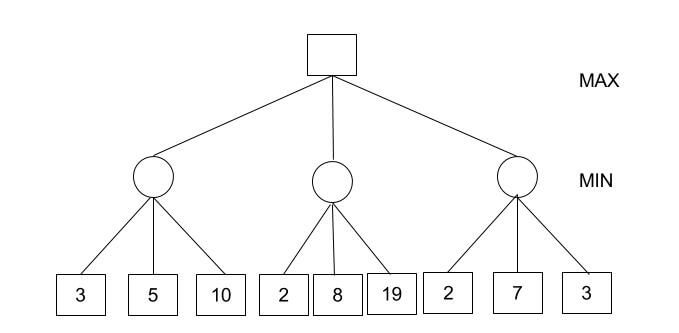
La question est maintenant de savoir si nous avons vraiment besoin de calculer c. Bien sûr que non.

Nous aurions pu arriver à une conclusion sans examiner ces nœuds. Et c’est là que l’élagage alpha-bêta entre en scène.

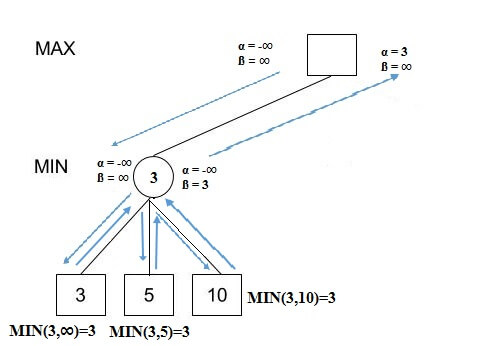
Quelques définitions :

* **Alpha:** C’est le meilleur choix jusqu’à présent pour le joueur MAX. Nous voulons obtenir la valeur la plus élevée possible ici.
* **Beta:** C’est le meilleur choix jusqu’à présent pour MIN, et il doit être la valeur la plus basse possible.
* **Node:** Chaque nœud doit garder une trace de ses valeurs alpha et bêta. L’alpha ne peut être mis à jour que lorsque c’est le tour de MAX et, de même, la version bêta ne peut être mise à jour que lorsque c’est le cas de MIN.

# 5-Comment fonctionne l’élagage alpha-bêta?

Initialisez alpha = -infinity et beta = infinity comme les pires cas possibles. La condition pour élaguer un nœud est lorsque alpha devient supérieur ou égal à bêta.[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/alpha-beta-pruning-1.jpg)

Commencez par assigner les valeurs initiales d’alpha et bêta à la racine et puisque alpha est inférieur à bêta, nous ne l’élaguons pas.

Portez ces valeurs alpha et bêta au nœud enfant sur la gauche. Et maintenant, à partir de la valeur d’utilité de l’état terminal, nous allons mettre à jour les valeurs de alpha et beta, de sorte que nous n’avons pas à mettre à jour la valeur de bêta. Encore une fois, nous ne taillons pas parce que la condition reste la même. De même, le troisième nœud enfant également. Et puis revenir à la racine, nous avons défini alpha = 3 parce que c’est la valeur minimale que l’alpha peut avoir.  
[](https://blog-c7ff.kxcdn.com/blog/wp-content/uploads/2017/03/Alpha-beta-1.jpg)

Maintenant, alpha = 3 et bêta = infini à la racine. Donc, nous n’élaguons pas. En portant cela au nœud central, et en calculant MIN{2, infini}, nous obtenons alpha=3 et beta=2.

Taillez les deuxième et troisième nœuds enfants car l’alpha est maintenant supérieur à bêta.

Alpha at the root remains 3 because it is greater than 2. Carrying this to the rightmost child node, evaluate MIN{infinity,2}=2. Update beta to 2 and alpha remains 3.

Prune the second and third child nodes because alpha is now greater than beta.

Hence, we get 3, 2, 2 at the left, center, and right MIN nodes, respectively. And calculating MAX{3,2,2}, we get 3. Therefore, without even looking at four leaves we could correctly find the minimax decision.

**Pseudocode :**

evaluate (node, alpha, beta)

if node is a leaf

return the utility value of node

if node is a minimizing node

for each child of node

beta = min (beta, evaluate (child, alpha, beta))

if beta <= alpha

return beta

return beta

if node is a maximizing node

for each child of node

alpha = max (alpha, evaluate (child, alpha, beta))

if beta <= alpha

return alpha

return alpha

# Conclusion

Les jeux sont très attrayants et écrire des programmes de jeu est peut-être encore plus excitant. Ce que les courses de Grand Prix sont à l’industrie automobile, le jeu l’est à l’IA.

Tout comme nous ne nous attendrions pas à ce qu’une voiture de course fonctionne parfaitement sur une route cahoteuse, nous ne devrions pas nous attendre à ce que les algorithmes de jeu soient parfaits pour chaque situation.

Il en va de même pour l’algorithme minimax. Ce n’est peut-être pas la meilleure solution à toutes sortes de jeux informatiques qui ont besoin d’IA.

Mais avec une bonne mise en œuvre, cela peut créer un concurrent difficile.

# Webographie

[Algorithme Minimax avec élagage alpha-bêta - HackerEarth Blog](https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/)