جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

ENSAM
المدرسة الوطنية العليا للفنون و المهن
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS

Tic-Tac-Toe en utilisant l'algorithme MinMax

Achraf *FAYTOUT*[1], Mourad *IZEM*[2]

Elèves ingénieurs Art et Métiers en Génie Industriel : Intelligence Artificielle et Data Science (GI-IADS)

Travail supervisé par : Dr Tarik HAJJI

Assistant Professor of Big Data and Artificial Intelligence à l'ENSAM-Meknès

Résumé

Tic-tac-toe is not a very challenging game for human beings. It is also a weak AI. The game play is very simple. Two players alternately put X's and O's in compartments of a figure formed by two vertical lines crossing two horizontal lines and each tries to get array of three X's or three O's before the opponent does. When you just look at the board and instantly know where you want to move. This kind of instant knowledge is great for human beings, because it makes you a fast player. But it isn't much help in writing a computer program. For that, you have to know very explicitly what your strategy is. In order to create a AI to play with a user. We have used Alpha Beta Pruning to achieve the goal state. Moreover, a minimax algorithm is also used. The minimax algorithm is like brute-force approach. It tries to see every possible outcome and then tries to optimize whatever options it has in hand.

Keywords : *IA, game, Tic-Tac-Toe MinMax, optimization.*

1 Introduction

Tic-tac-toe également connu sous le nom de « noughts and crosses »est un jeu de papier/crayon pour deux joueurs, qui prennent tours en marquant les espaces dans une grille 3 x 3. Le joueur qui réussit à placer trois de ses marques dans une rangée horizontale, verticale ou diagonale gagne la partie. Ce jeu est caractérisé par un environnement déterministe, totalement observable, dans lequel deux agents agissent alternativement et où les valeurs d'utilité à la fin du jeu sont toujours égales et opposées. En raison de la simplicité du tic-tac-toe, il est souvent utilisé comme outil pédagogique en intelligence artificielle pour se familiariser avec la recherche d'arbres de jeu. Le mouvement optimal pour ce jeu peut être obtenu en utilisant l'algorithme minimax nécessitant généralement l'alpha bêta pruning d'intelligence artificielle.

1.1 Objectifs

1. Développer un jeu tic-tac-toe basé sur l'intelligence artificielle pour l'homme Vs IA en implémentant l'algorithme minimax.
2. Analyser la complexité de l'algorithme minimax à travers un jeu de dimension supérieure à 3 (cas 4x4).
3. Étudier et mettre en œuvre le concept alpha-bêta pruning pour une vitesse de recherche améliorée.

1.2 Agent

Un agent perçoit son environnement à l'aide de capteurs et agit sur lui en utilisant des actionneurs. Son comportement est décrit par sa fonction qui mappe le percept à l'action. Dans le tic tac toe, il y a deux agents, l'ordinateur et l'humain. Dans ce jeu basé sur l'IA, la fonction est cartographiée à l'aide de l'algorithme minimax. L'agent peut être décrit plus en détail par les facteurs suivants :

- Mesure du rendement : Nombre de victoires ou de tirages.
- Environnement : Une grille 3x3 ou plus, avec adversaire (agent humain). La cellule autrefois

occupée par une marque ne peut plus être utilisée. L'environnement est
déterministe, entièrement observable, statique, multi-agents, discret, séquentiel.

— Actionneurs : Affichage (Screen), Souris (agent humain)
— Capteurs : L'entrée de l'adversaire (agent humain) en tant qu'un de pointeur de souris sur une cellule.

En outre, il s'agit d'un agent basé sur l'utilité puisqu'il utilise l'heuristique et le concept d'élagage. La fonction d'utilité mesure ses préférences parmi les états possibles et choisit l'action qui conduit à le meilleure résultat attendu, c'est-à-dire la condition gagnante ou d'égalité pour l'agent informatique.

1.3 L'environnement de l'agent

L'environnement de base pour l'agent de jeu tic tac toe est une grille 3x3 avec 9 cellules respectivement avec l'adversaire comme agent humain. La cellule autrefois occupée par une marque ne peut plus être utilisée. L'agent a accès à l'état complet de l'environnement à tout moment et peut détecter tous les aspects pertinents pour le choix de l'action, ce qui rend l'environnement entièrement observable. En outre, l'état suivant de l'environnement peut être complètement déterminé par l'état actuel et l'action exécutée, ce qui signifie que l'environnement est déterministe. C'est un jeu à deux joueurs avec l'adversaire comme agent humain, c'est donc un environnement multi-agents. A côté du jeu, l'environnement a un nombre fini d'états, de percepts et d'actions, ce qui rend l'environnement statique et discret. De plus, l'environnement est séquentiel puisque le choix actuel de la cellule pourrait affecter toutes les décisions futures. Ainsi, l'environnement de l'agent système tic-tac toe est séquentiel, déterministe, entièrement observable, statique, discret et multi-agent.

2 Spécification du problème de dimension supérieure à 3 (cas 4x4)

Le jeu Tic tac toe de dimension 4 dispose de 16 cellules pour une grille 4x4. Les deux joueurs avec leurs marques respectives comme 'X' et 'O' sont tenus de placer leurs marques dans leurs tours un par un. Une fois que la cellule est occupée par une marque, elle ne peut plus être utilisée. La partie est gagnée si l'agent est capable de faire une ligne ou une colonne ou une diagonale occupée complètement avec leurs marques respectives. Le jeu se termine une fois que la situation gagnante est atteinte ou que les cellules sont entièrement occupées. La spécification du problème 4X4 est donnée ci-dessous :

Problème : Étant donné une grille 4x4, les agents doivent trouver la cellule optimale à remplir avec des marques respectives.
Objectifs : Pour trouver la cellule optimale à remplir avec les notes respectives et pour gagner la partie, la cellule doit être remplie de manière à satisfaire à l'un des critères suivants :

1. Une ligne est complètement remplie par une marque 'X' ou 'O'.
2. Une diagonale est complètement remplie par une marque 'X' ou 'O'.
3. Une colonne est complètement remplie par une marque 'X' ou 'O'.
   Si ces critères ne sont pas satisfaits par les deux agents, le jeu se termine par une situation d'égalité

   Contraintes :

   (a) Une fois que la cellule est occupée par une marque, elle ne peut pas être réutilisée.
   (b) Les agents placent la marque alternativement. Ainsi, les mouvements consécutifs de n'importe quel agent ne sont pas autorisés.

2.1 Espace des états

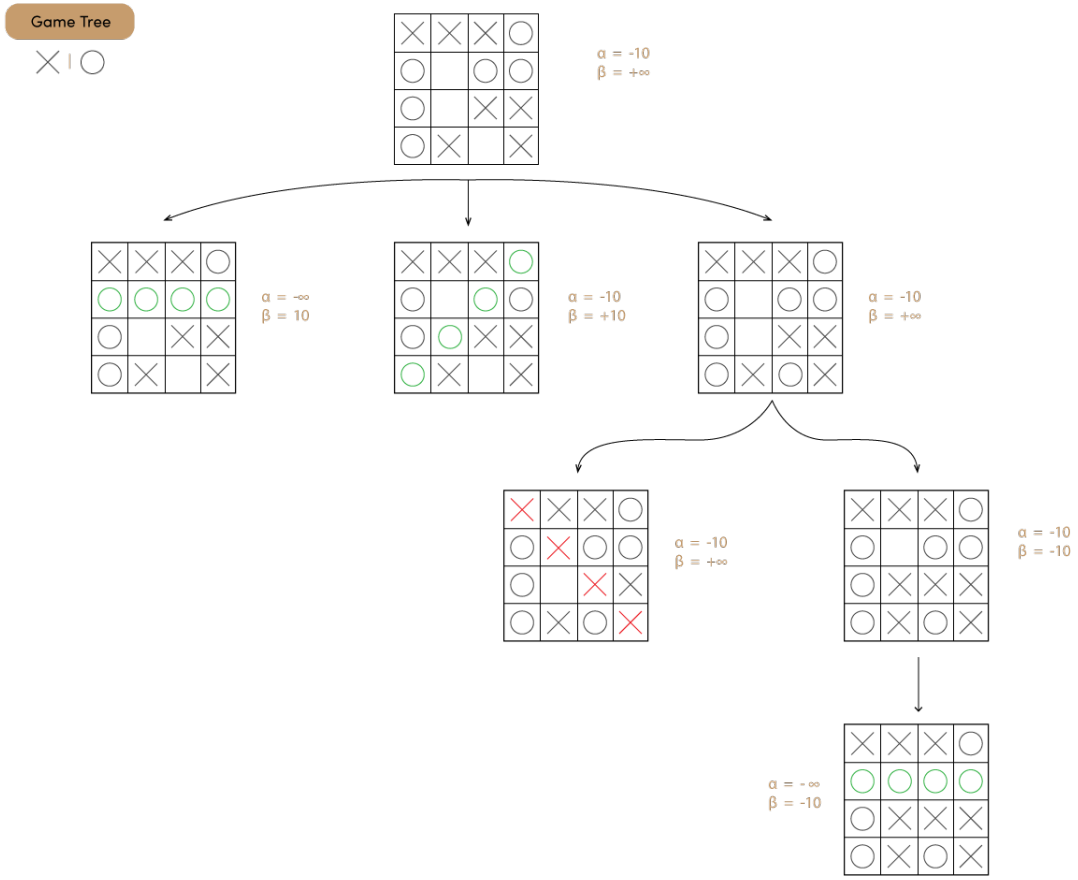

FIGURE 1 – représentation de l'espace d'état

3 Algorithme exploité

3.1 Algorithme Minimax

Minimax est un algorithme récursif qui est utilisé pour choisir un mouvement optimal pour un joueur en supposant que l'adversaire joue également de manière optimale. L'algorithme s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle (les gains réalisés par un joueur sont des pertes pour l'autre joueur) et à information complète. Son objectif est de minimiser la perte maximale.
On va noter par MAX l'agent qu'on cherche à faire gagner et son adversaire par MIN. Les deux joueurs désirent gagner le jeu. On suppose que le joueur MIN joue logiquement et qu'il ne va jamais rater une occasion de gagner. Si pour gagner le joueur MAX essaie de maximiser son score, le joueur MIN désire aussi maximiser son propre score (ou de minimiser le score du joueur MAX). À chaque tour le joueur MAX va choisir le coup qui va maximiser son score, tout en minimisant les bénéfices de l'adversaire. Ces bénéfices sont évalués en termes de la fonction d'évaluation statique utilisée pour apprécier les positions pendant le jeu.
l'évaluation statique des positions terminales de jeu est simple et précise : il faut apprécier selon les règles du jeu s'il s'agit d'une victoire, d'une défaite ou d'une égalité. Pour les postions intermédiaires cette fonction est imprécise, à cause des critères plus au moins subjectifs qui ont été utilisé pour l'évaluation. Le manque d'exactitude de la fonction d'évaluation statique est compensée par l'analyse rigoureuse des conséquences de chaque coup de deux joueurs.
L'algorithme réalise une évaluation de la position courante, représentée par la racine de l'arbre de jeu, en partant des nœuds terminaux. Dans ce but, l'ensemble des nœuds de l'arbre de jeu est divisé en deux classes : les nœuds MAX sur les niveaux où ce joueur va choisir un coup et les nœuds MIN pour les niveaux de décision du joueur MIN.
l'algorithme se caractérise par les performance suivantes :
- Complètude – oui si l'arbre de jeu est fini.
- Optimalité – oui si les deux joueurs jouent de façon optimale.
- Complexité en temps - $O(b^m)$
- Complexité en espace – pour une exploration de l'arbre de jeu en profondeur d'abord $O(bm)$

3.2 Élagage alpha-bêta

Le problème avec la recherche minimax est que le nombre d'états de jeu qu'il doit examiner est exponentiel dans la profondeur de l'arbre. L'algorithme minimax s'appelle récursivement jusqu'à ce que l'un des agents gagne ou que la carte soit pleine, ce qui prend beaucoup de temps de calcul et rend impossible la résolution de la grille 4X4 à l'aide de l'algorithme minimax classique.
Pour résoudre ce problème, nous pouvons utiliser un algorithme d'élagage alpha-bêta qui élimine de grandes parties de l'arbre des considérations en élaguant l'arbre. Lorsqu'il est appliqué à un arbre minimax standard, il retourne le même mouvement que minimax, mais il élague les branches qui ne peuvent pas influencer la décision finale. Fondamentalement, cet algorithme applique le principe selon lequel il ne sert à rien de dépenser du temps de recherche pour savoir exactement à quel point une alternative est mauvaise, si vous avez une meilleure alternative. L'élagage alpha-bêta tire son nom des paramètres liés aux valeurs sauvegardées qui apparaissent n'importe où le long du chemin :
α=la valeur des meilleurs choix (c'est-à-dire la valeur la plus élevée) jusqu'à présent à n'importe quel point de choix le long du chemin pour MAX.
β= la valeur du meilleur choix (c'est-à-dire la valeur la plus basse) jusqu'à présent à n'importe quel point de choix le long du chemin pour MIN.

4 Interprétation des résultats

L'algorithme minimax effectue une exploration complète de l'arbre de jeu en profondeur. Il s'appelle récursivement jusqu'à ce qu'un état gagnant d'un agent soit trouvé ou que la grille soit pleine. Si la profondeur maximale de l'arbre est m et qu'il y a b mouvements légaux en chaque point, alors la complexité temporelle de l'algorithme minimax est $O(b^m)$. La complexité de l'espace est O(bm) pour générer toutes les actions à la fois. Pour la carte 3X3, le nombre possible de mouvements est de $(3 * 3)! = 9!$ Si la taille de la carte est augmentée à 4X4, il y a une croissance exponentielle et nous connaissons une explosion du temps de calcul. Cela signifie que nous en avons maintenant $16!$ États possibles , ce qui est une quantité incroyablement importante. Ainsi, nous pouvons estimer que le temps de calcul sera des millions de fois supérieur à celui de la carte 3X3. Par conséquent, le coût en temps est totalement irréalisable, ce qui rend cet algorithme irréalisable en 4X4, mais il sert de base à l'analyse mathématique des jeux et à un algorithme plus pratique.
Pour implémenter l'algorithme minimax pour la carte 4X4, nous avons appliqué le concept d'élagage alpha bêta pour éliminer les solutions qui n'auraient pas d'impact sur la décision finale. En considérant les paramètres ci-dessus, la complexité temporelle ou l'élagage alpha-bêta est $O(b^m/2)$ dans le meilleur des cas. C'est-à-dire que la profondeur est réduite de moitié dans le meilleur scénario possible. Même s'il y a une telle réduction du temps, la complexité du temps est encore très grande compte tenu de notre conseil d'administration. Ainsi, pour diminuer le temps de calcul, la recherche peut être limitée à un certain niveau, c'est-à-dire réduire la profondeur de l'arbre de recherche. La profondeur maximale est prise à 9 dans notre système. Mais en réduisant la profondeur de l'arbre de recherche, la sortie n'est pas optimale car nous avons coupé 9 niveaux de notre arbre de recherche qui est un très grand nombre de nœuds. Par conséquent, nous avons utilisé une fonction d'évaluation heuristique qui se rapproche de l'utilité réelle d'un état sans effectuer de recherche complète. Pour ce problème, la fonction heuristique sur utilisé est donné par :

$$E(n) = M(n)–O(n)$$

Où, $M(n)$ est le total de la voie gagnante possible de l'IA.
$O(n)$ est le total du chemin de victoire possible du joueur humain $E(n)$ est l'évaluation totale pour l'état n.
Cette heuristique donne une valeur positive si l'IA, c'est-à-dire maximiser l'agent, a plus de chances ou de gagner et une valeur négative si le joueur humain, c'est-à-dire l'agent minimisant, a plus de chances de gagner.

5 Conception de l'agent et de l'environnement

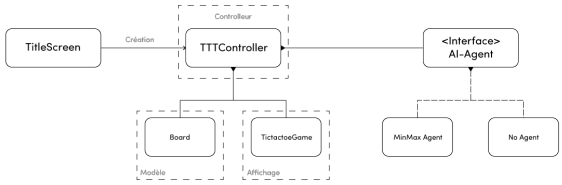

FIGURE 2 – Diagramme des classes

Pour les composants du système, on trouve :
— **TitleScreen** [Point d'entrée] : Cette classe est responsable de l'interface graphique de l'écran de titre, sa fonctionnalité principale est de rassembler les options de jeu définies par l'utilisateur, puis de démarrer un jeu en créant un contrôleur Tic Tac Toe avec les options de l'utilisateur.
— **TTTController** [contrôleur] : Cette classe est la manette d'un jeu Tic Tac Toe, elle prend en entrée les options du jeu, et contrôle la vue (TicTacToeGame) et le modèle (Board) selon la logique de jeu spécifiée dans cette classe, c'est aussi chargé d'appeler l'agent IA pour jouer son tour si spécifié par les options du jeu.
— **Board** [Modèle] : cette classe est chargée de représenter le modèle sous-jacent du jeu. Il s'agit

d'une représentation matricielle 2D de l'interface graphique lisible par machine et pouvant être utilisée lors de l'exécution de tout type d'algorithmes sur la carte.

— **TicTacToeGame** [View] : Cette classe est responsable de l'interface graphique d'un jeu, elle est responsable de la construction des cellules d'un tableau, qui peut être de taille variable, et d'un bouton de navigation pour ramener l'utilisateur à l'écran titre, et d'un bouton pour réinitialiser le jeu et en jouer un nouveau.

— **AIAgent** [Interface] : cette interface décrit le comportement de tous les agents IA implémentés.

— **MiniMaxAgent** [Implementation de l'agent IA] : Cette classe gère toute la logique de sélection d'une cellule basée sur l'algorithme MiniMax.

— **NoAgent** [Implementation de l'agent IA] : Cette classe est une classe vide qui ne joue pas, elle spécifie le comportement de l'objet nul, elle représente donc l'absence d'un agent IA, ce qui est utile pour éviter d'utiliser des conditions if dans le code lors de la vérification de l'existence d'un agent IA.

Concernat le flux du systéme, Le programme commence à collecter les options de jeu en utilisant la classe TitleScreen, lorsque l'utilisateur clique sur "Start game", cette classe crée un TTTController qui gérera un jeu Tic Tac Toe, il créera un modèle (Board) et une vue (TicTacToeGame) et spécifiez le type d'agent AI (MiniMax ou NoAgent), lorsqu'un utilisateur clique sur une cellule, le contrôleur indique à la vue de mettre à jour l'apparence de la cellule cliquée en définissant la marque du joueur (X ou O), et il indique au tableau de mettre à jour sa représentation sous-jacente du tableau, puis il appelle la fonction évalueBoard de la classe du tableau pour voir si nous avons atteint un état final (Quelqu'un a gagné ou Égalité), s'il s'agit d'un état final, il déclare l'état via une QMessageBox.

Parmis les caractéristique de notre jeu, on cite :

— Taille de plateau dynamique, choisissez n'importe quelle taille de plateau pour jouer.

— Jouez contre un agent MiniMax AI avec élagage alpha bêta, ou jouez à un jeu avec deux joueurs à tour de rôle.

— Spécifiez si l'IA démarre en premier ou si vous démarrez en premier.

— Spécifiez la profondeur de coupure Minimax pour contrôler la difficulté de l'agent MiniMax.

— L'augmentation de la profondeur affectera les performances.

FIGURE 3 – L'interface "titleGame"

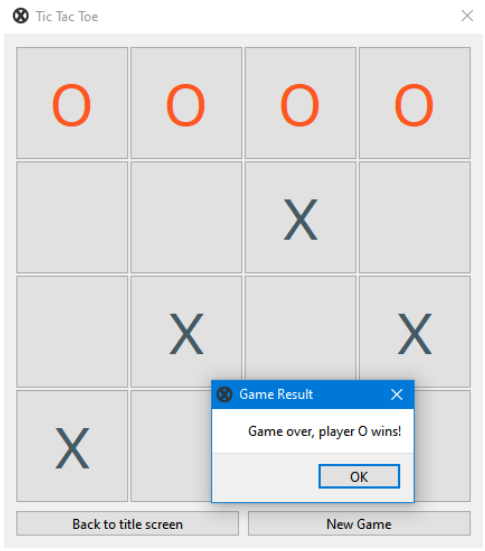

FIGURE 4 – L'interface "tictactoeGame"

6 Conclusion

Avec la base de l'algorithme minimax pour l'analyse mathématique, ainsi que l'accélération du calcul par le concept d'élagage alpha bêta et l'optimisation de la fonction utilitaire,Nous avons exploré qu'un 4x4 tic tac toe, un jeu de technique de recherche d'adversaire en intelligence artificielle, peut être développé. Or, L'augmentation de la taille de ce jeu créerait un énorme problème de complexité temporelle avec le même algorithme et les mêmes techniques, pour lesquels d'autres logiques doivent être étudiées.

Références

[1] *Peter Baum.* [Paper]. "Tic Tac Toe"

[2] *Swaminathan.B , Vaishali.R and subashri.T* [Paper]. "Analysis of Minimax Algorithm Using Tic-Tac-Toe"

[3] *Shahd H. Alkaraz, Essam El-Seidy, Neveen S. Morcos* [Paper]. "Tic-Tac-Toe : Understanding the Minimax Algorithm"

[4] *Zain AM1, Chai CW , Goh CC , Lim BJ , Low CJ , Tan SJ* [Paper]. "Development of Tic-Tac-Toe Game Using Heuristic Search"

[5] *K. Kask.* [En ligne]. https ://www.ics.uci.edu/ kkask/Fall- 2016

[6] *G. Surma.* [En ligne]. https ://towardsdatascience.com/tic-tac-toe-creating-unbeatable-ai-with-minimax-algorithm-8af9e52c1e7d

[7] *Sunil Karamchandani.* [Livre]. ≪ A Simple Algorithm For Designing An Artificial Intelligence Based Tic Tac Toe Game ≫.

[8] *edureka !.* [En ligne]. https ://www.edureka.co/blog/alpha-beta-pruning-in-ai.

Notre vidéo de présentation

`https://youtu.be/JfNR6zEKbc4`