ANNÉE 2020 - 2021

1921.1 Communication — Rapport technique — ISC1D

# Intelligence artificielle

## Implémenter une intelligence artificielle pour le jeu du morpion avec l'algorithme Minimax

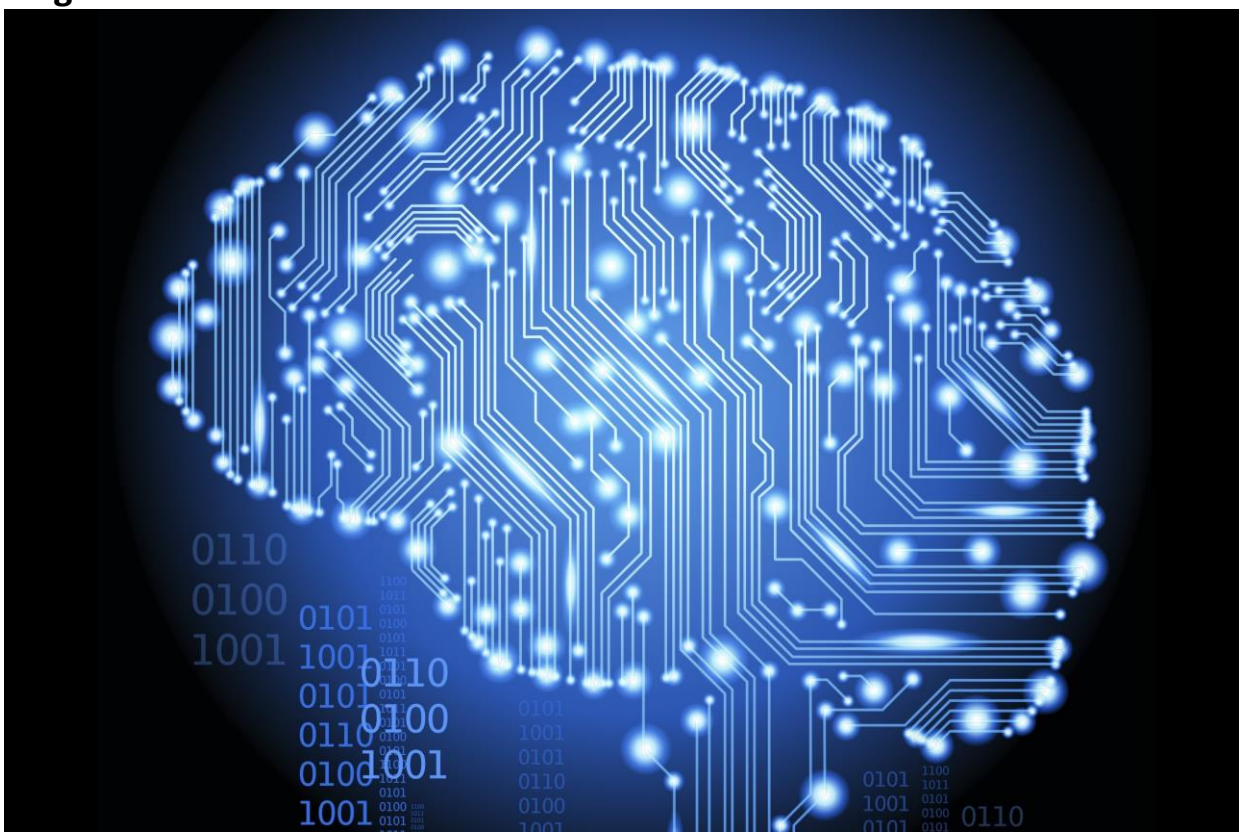

*Figure 1: Image de page de garde représentant l'intelligence artificielle*


Étudiants participant à ce travail :

**Nicolas AUBERT, ISC1D**
**Noël DRUART, ISC1D**

Présenté à :

**Raboud Stéphane**


Restitution du rapport : **22.02.2021**

Période : **2020 – 2021**


École : **HE-Arc, Neuchâtel**

## Abstract

Ce rapport présente le fonctionnement d'une intelligence artificielle basée sur l'algorithme Minimax, ainsi que son implémentation sur le jeu du morpion. Il se joue sur une grille de trois par trois cases, sur laquelle deux joueurs doivent placer à tour de rôle leur symbole (« X » et « O »). Afin de gagner la partie, un des joueurs doit former une ligne (horizontale, verticale, diagonale) avec trois de ses symboles.

L'intelligence teste à chaque tour toutes les possibilités engendrées par la disposition actuelle, et attribue un score à chacune d'entre-elles. Elle parcourt ensuite les différentes positions de jeu dans le sens inverse, des dernières aux premières. Sur les lignes paires, elle conserve la disposition possédant le score le plus élevé. Sur les lignes impaires, elle en fait de même, mais avec la disposition possédant le score le plus faible. Ce procédé permet alors de déterminer le choix le plus judicieux afin de remporter la partie. Il est donc impossible que l'intelligence soit vaincue.

# Table des matières

**1 - LISTES DIVERSES ..... 1**

1.1 - Liste des figures ..... 1
1.2 - Glossaire ..... 1

**2 - INTRODUCTION ..... 2**

2.1 - Problématique ..... 2
2.2 - Objectifs du rapport ..... 2
2.3 - Méthodologie ..... 2

**3 - LE JEU DU MORPION ..... 3**

3.1 - Fonctionnement du jeu ..... 3
*3.1.1 - Règle du jeu ..... 3*
*3.1.2 - Condition de victoire ..... 3*
*3.1.3 - Égalité ..... 3*
3.2 - Implémentation du jeu en C# ..... 4
*3.2.1 - Plateau de jeu / Affichage des joueurs ..... 4*
*3.2.2 - Condition de victoire ..... 4*

**4 - ALGORITHME MINIMAX ..... 5**

4.1 - Fonctionnement de l'algorithme ..... 5
*4.1.1 - Maximisation et minimisation ..... 6*
4.2 - Minimax avec le morpion ..... 7
*4.2.1 - Calculs des scores ..... 7*
*4.2.2 - Maximisation et minimisation ..... 8*
4.3 - Implémentation de l'algorithme dans le jeu ..... 9
*4.3.1 - Récupération du meilleur mouvement ..... 9*
*4.3.2 - Fonction de minimisation ..... 9*
*4.3.3 - Fonction de maximisation ..... 9*
4.4 - Résultats ..... 10
*4.4.1 - Exemple de partie où l'IA joue en premier ..... 10*
*4.4.2 - Exemple de partie où l'IA joue en deuxième ..... 11*

**5 - POUR ALLER PLUS LOIN ..... 13**

5.1 - Simplification Negamax ..... 13
5.2 - Élagage alpha-bêta ..... 13
5.3 - Machine learning ..... 14

**6 - CONCLUSION ..... 15**

**7 - BIBLIOGRAPHIES ET RÉFÉRENCES ..... 16**

*7.1.1 - Sites Internet ..... 16*

**8 - ANNEXES ..... 17**

8.1 - Code source du morpion avec Minimax ..... 17

# 1 - Listes diverses

## 1.1 - Liste des figures

*Figure 1:* Image de page de garde représentant l'intelligence artificielle.
*Source : https://urlz.fr/eYfM*

*Figure 2* : Exemples de combinaisons gagnantes au morpion.

*Figure 3* : Exemple de parties nulles au morpion.

*Figure 4* : Démonstration du programme « morpion » réalisé en C#.

*Figure 5* : Exemple d'un schéma en arbre sur lequel se base l'algorithme Minimax.

*Figure 6* : Exemple d'un schéma en arbre avec du jeu du morpion.

*Figure 7* : Exemple d'un schéma en arbre simplifié avec les attributions de scores par l'algorithme Minimax.

*Figure 8* : Démonstration d'une partie où l'IA joue en premier — Tour un à trois.

*Figure 9* : Démonstration d'une partie où l'IA joue en premier — Tour quatre à sept, victoire de l'IA.

*Figure 10* : Démonstration d'une partie où l'IA joue en deuxième — Tour un à deux.

*Figure 11* : Démonstration d'une partie où l'IA joue en deuxième — Tour trois à six.

*Figure 12* : Démonstration d'une partie où l'IA joue en deuxième — Tour sept à huit, égalité.

*Figure 13* : Exemple de schéma en arbre avec l'élagage alpha-bêta

## 1.2 - Glossaire

| | |
|---|---|
| **Algorithme** | Suite de règles auxquelles un programme informatique doit obéir. |
| **C#** | Langage de programmation orienté objet plutôt universel permettant de créer des applications sur différentes plateformes. |
| **Intelligence artificielle (IA)** | Technologie permettant à une machine de « réfléchir » par elle-même. |
| **Machine learning** | Intelligence artificielle apprenant par elle-même (à jouer à un jeu par exemple). |
| **Récursion** | Répétition d'une partie de code dans un programme. Par exemple, le fait de pouvoir s'appeler soi-même à l'intérieur d'une fonction et donc de créer une boucle. |
| **Connivence** | Autoréflexion, prise de décision autonome et non programmée. |

# 2 - Introduction

Ce rapport est réalisé dans le cadre du cours de Communication 1921.1, dispensé par M. Raboud Stéphane. Il nous a demandé de choisir un thème, puis de l'étudier afin de trouver une problématique liée. Par la suite, nous avons dû écrire un rapport sur cette problématique dans le but d'y répondre. L'idée vers laquelle nous nous sommes orientés est « L'intelligence artificielle ».

## 2.1 - Problématique

**Comment créer une intelligence artificielle capable de jouer au morpion ?**

Le principe d'une intelligence artificielle (IA) est de pouvoir simuler les fonctions mentales et la connivence. De ce point de vue, il est normal de se poser la question « Comment est-ce que cela fonctionne ? ». C'est ce à quoi nous allons répondre au cours de ce rapport, en illustrant ces propos avec un exemple concret : le jeu du morpion.

## 2.2 - Objectifs du rapport

L'objectif premier consiste à créer une réplique du jeu du morpion, à l'aide du langage de programmation C#. Le second, qui est aussi le principal, englobe la conception d'une intelligence artificielle capable de jouer à ce jeu, et contre laquelle il n'est pas possible de gagner.

Ce rapport contiendra alors toutes les informations nécessaires afin d'y parvenir.

## 2.3 - Méthodologie

Premièrement, nous allons rechercher des informations sur des algorithmes permettant de jouer à des jeux simples, au coup par coup et à deux compétiteurs : comme le morpion, les dames ou encore les échecs.

Une fois que nous saurons comment implémenter une telle intelligence artificielle, nous allons créer la réplique du morpion en C#. Cette partie ne devrait pas poser de problème majeur : le C# est un langage de programmation que nous maîtrisons.

Dès que le jeu sera fonctionnel, nous entamerons l'implémentation de l'algorithme afin de concevoir l'intelligence artificielle. Nous testerons ensuite son bon fonctionnement.

# 3 - Le jeu du morpion

## 3.1 - Fonctionnement du jeu

### *3.1.1 -Règle du jeu*

Le jeu se joue sur un plateau de trois par trois cases. Le premier et le deuxième joueur se voient assigner un symbole, respectivement le « X » et le « O ». Chaque compétiteur place à tour de rôle son symbole sur une case vide du plateau.

### *3.1.2 -Condition de victoire*

Si un des joueurs arrive à former un alignement avec trois de ses symboles, sur une ligne, une colonne ou une diagonale, il gagne la partie.

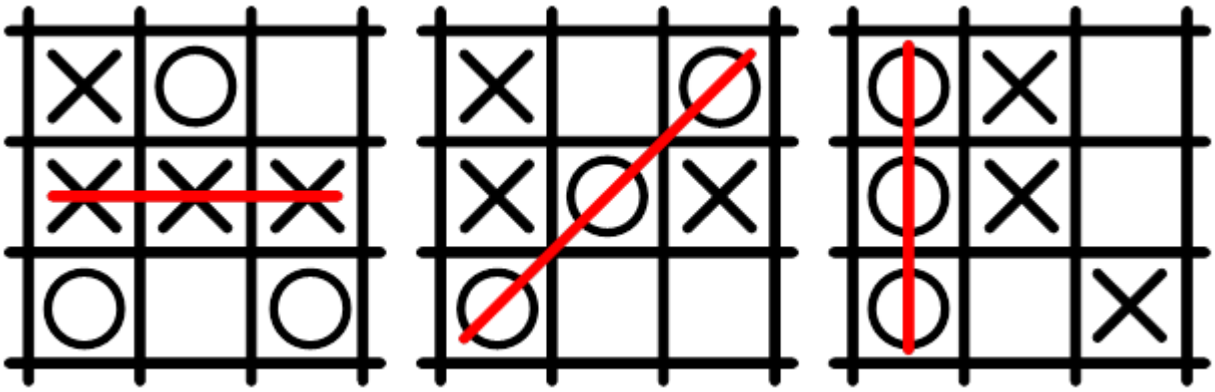

*Figure 2 : Exemples de combinaisons gagnantes au morpion*

### *3.1.3 -Égalité*

Si le plateau est rempli et qu'aucun alignement n'a été formé, personne ne gagne, comme illustré ci-dessous.

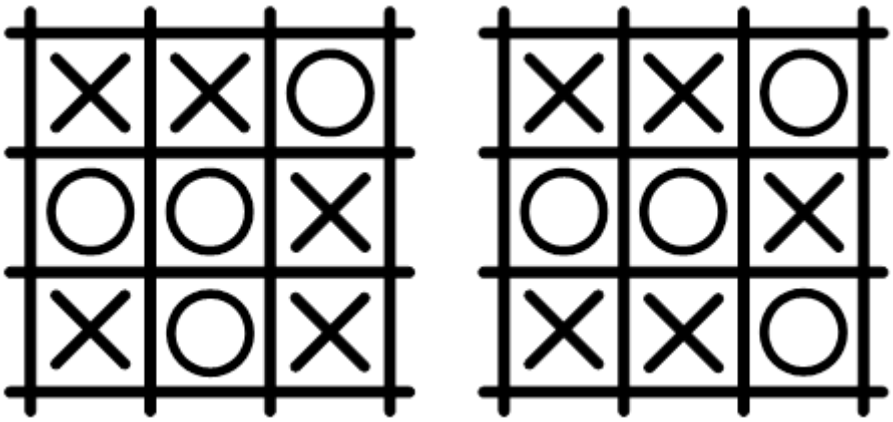

*Figure 3 : Exemple de parties nulles au morpion*

## 3.2 - Implémentation du jeu en C#

Pour simplifier la création de ce jeu, nous avons décidé de l'implémenter avec le langage de programmation C#, sur lequel nous possédons des connaissances poussées. Il a été réalisé en Windows Forms, grâce à l'IDE Visual Studio Code 2017.

### *3.2.1 -Plateau de jeu / Affichage des joueurs*

Le plateau de jeu est formé de trois colonnes de trois boutons. Chaque bouton est relié à un évènement lors d'un clic. Cet évènement permet d'afficher le symbole « X » ou « O » (en fonction du joueur qui possède le tour) sur la case cliquée.

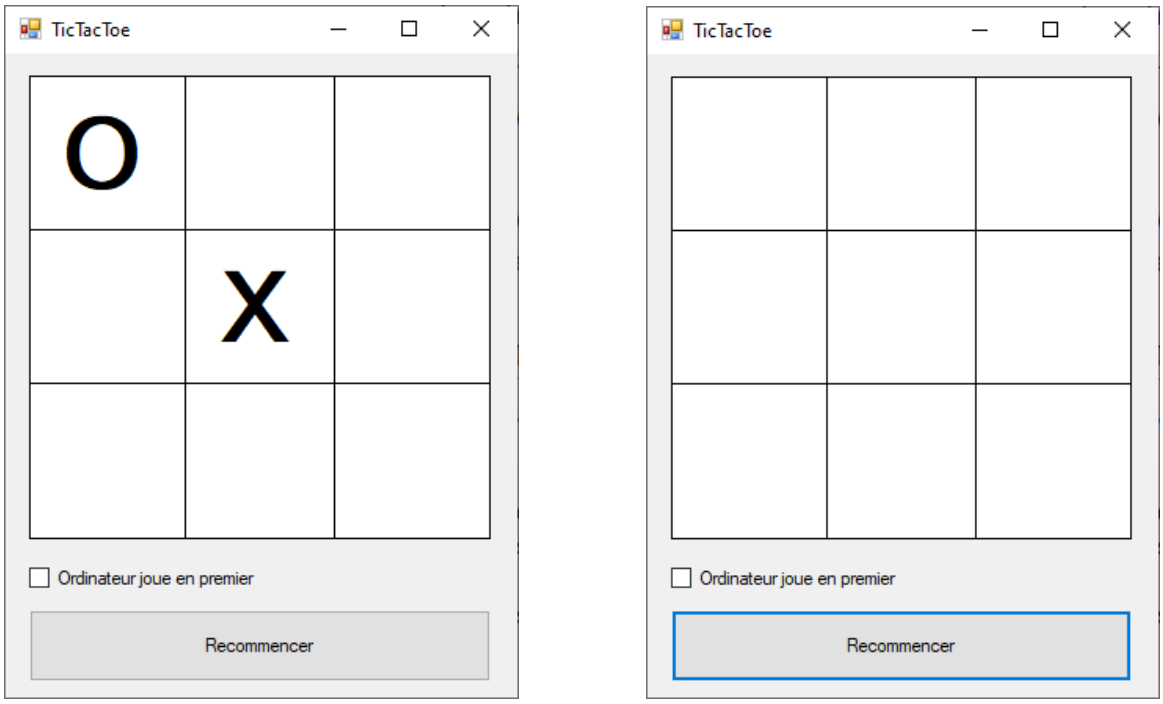


*Figure 4 : Démonstration du programme « morpion » réalisé en C#*

Le contenu affiché sur les boutons n'est pas utilisé pour calculer les conditions de victoire. C'est pourquoi le contenu du plateau est également mis à jour dans une matrice de trois par 3. Cela simplifie les manipulations pour les calculs.

Le plateau dispose également d'une case à cocher « Ordinateur joue en premier » permettant de choisir si l'IA joue en premier ou en deuxième.

### *3.2.2 -Condition de victoire*

À chaque nouveau tour, une fonction contrôle si une des conditions de victoire est remplie. Si c'est le cas, la partie est arrêtée, et un message affiche le joueur gagnant.

Cette fonction contrôle si chaque ligne, chaque colonne et chaque diagonale contient trois fois le même symbole. Si c'est le cas, elle retourne le symbole du joueur gagnant, sinon elle ne retourne rien.

# 4 - Algorithme Minimax

## 4.1 - Fonctionnement de l'algorithme

L'algorithme Minimax consiste à minimiser la perte maximum dans des jeux à deux joueurs (à somme nulle), comme le morpion, les dames, les échecs, etc. Il consiste à calculer toutes les possibilités de jeu et à assigner des valeurs à chaque possibilité en fonction des bénéfices pour le joueur courant et pour l'adversaire.

Cet algorithme peut être représenté avec un diagramme en arbre comme affiché ci-dessous.

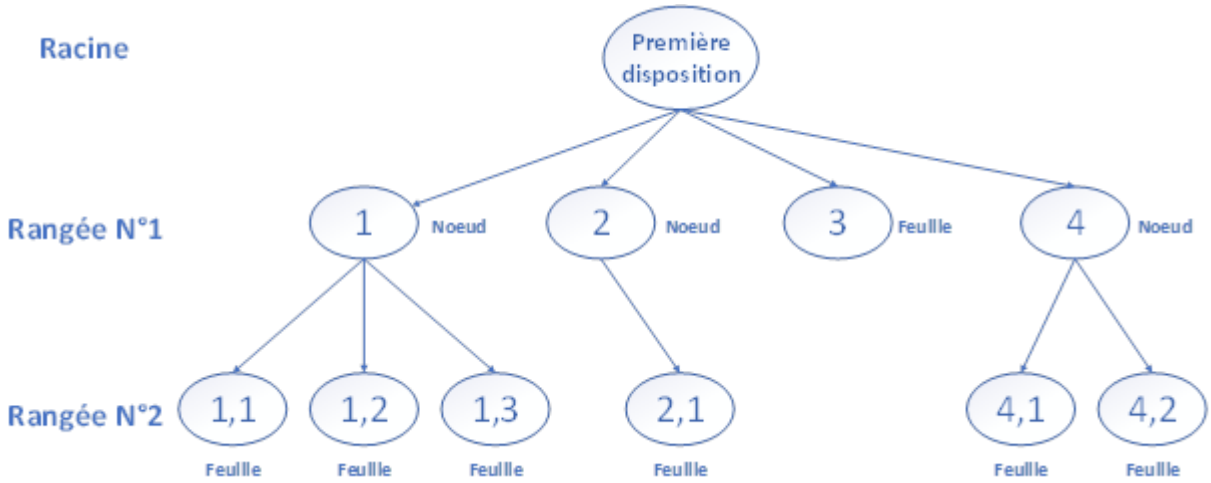


*Figure 5 : Exemple d'un schéma en arbre sur lequel se base l'algorithme Minimax*

Le premier nœud représente la disposition initiale, qui est la racine de l'arbre (le début). Elle se trouve en haut du diagramme. Sur la rangée numéro un se trouvent toutes les possibilités engendrées par le premier nœud (première disposition). Il en va de même pour les rangées suivantes où se trouvent les dispositions engendrées par les nœuds de la rangée précédente.

Chaque nœud qui ne donne pas suite à d'autres nœuds, appelés feuille, par exemple si la partie est terminée, est affecté d'un score. Une fois toutes les possibilités calculées et toutes les feuilles affectées par un score, l'algorithme parcourt l'arbre de bas en haut afin de faire remonter un score à la racine. Ce score représente alors le meilleur choix à faire par rapport à la situation initiale. Il est calculé récursivement avec les fonctions de maximisation et de minimisation suivantes :

### *4.1.1 -Maximisation et minimisation*

Le premier joueur maximise son score, alors que le deuxième (l'opposant) le minimise.

Lors de la maximisation, les règles suivantes sont obéies :

- Si le nœud est une feuille :
    - Lui attribuer un score
    - Donner ce score à la fonction de **minimisation**
- Si le nœud n'est pas une feuille :
    - Obtention du score par la fonction de **minimisation**,
    - Si ce score est **meilleur** que le précédent :
        - Conserver le score

Lors de la minimisation, les règles suivantes sont obéies :

- Si le nœud est une feuille :
    - Lui attribuer un score
    - Donner ce score à la fonction de **maximisation**
- Si le nœud n'est pas une feuille :
    - Obtention du score par la fonction de **maximisation**,
    - Si ce score est **moins bon** que le précédent :
        - Conserver le score

Afin de rendre ces explications un peu plus claires, plaçons-les dans un contexte et prenons le jeu du morpion comme exemple :

## 4.2 - Minimax avec le morpion

### *4.2.1 -Calculs des scores*

Voici un exemple d'un morceau de l'arbre, à partir d'une disposition de jeu assez avancée, afin de limiter le nombre de possibilités à représenter sur le schéma. Les grilles bleues signifient que le jeu est terminé (feuilles de l'arbre). Ces grilles se verront attribuer un score (affiché sous la grille, en bleu) :

- +10 si « X » gagne,
- 0 en cas d'égalité,
- -10 si « O » gagne.

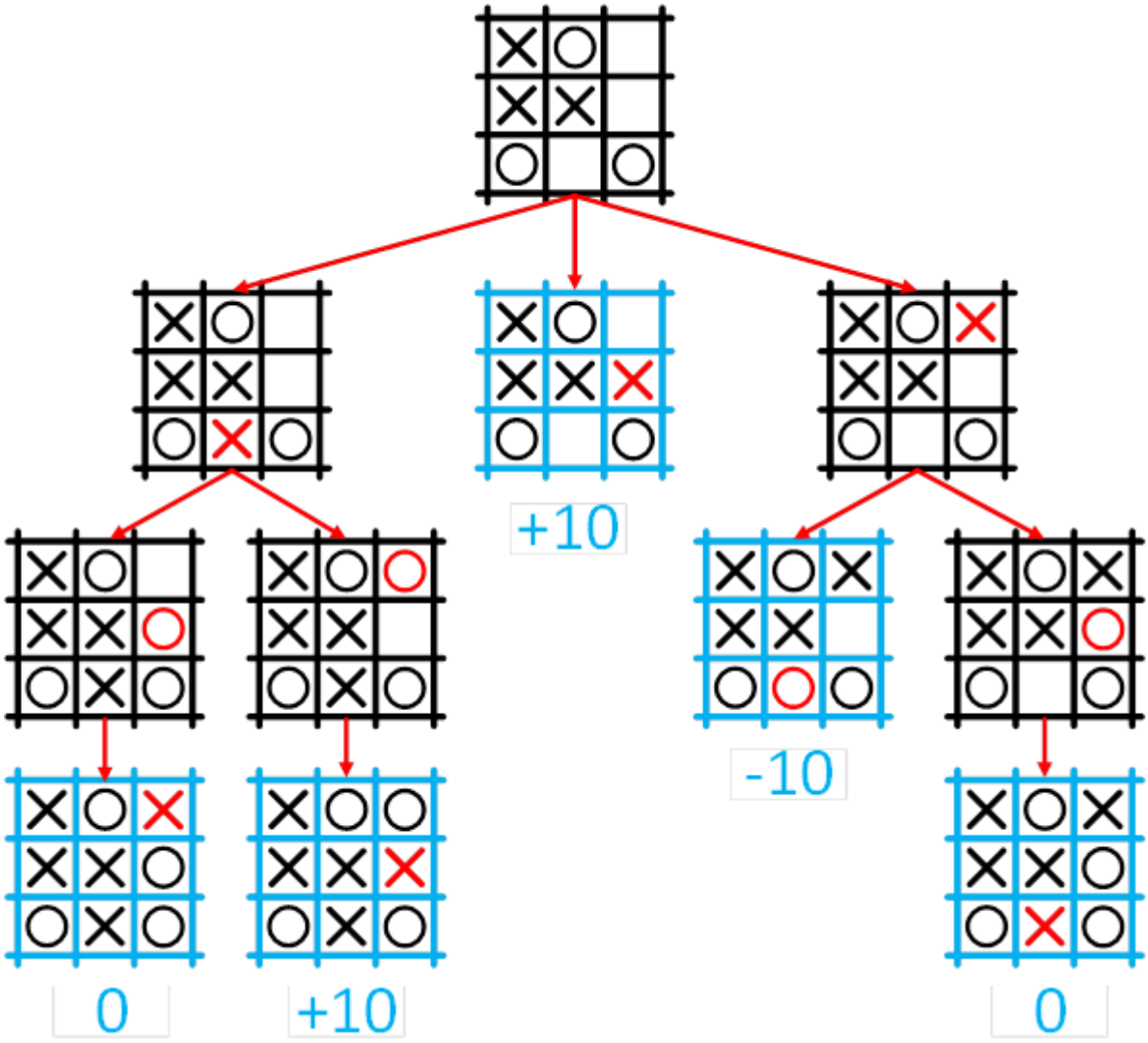


*Figure 6 : Exemple d'un schéma en arbre avec du jeu du morpion*

### *4.2.2 -Maximisation et minimisation*

Ce schéma représente le même arbre qu'au point précédent.

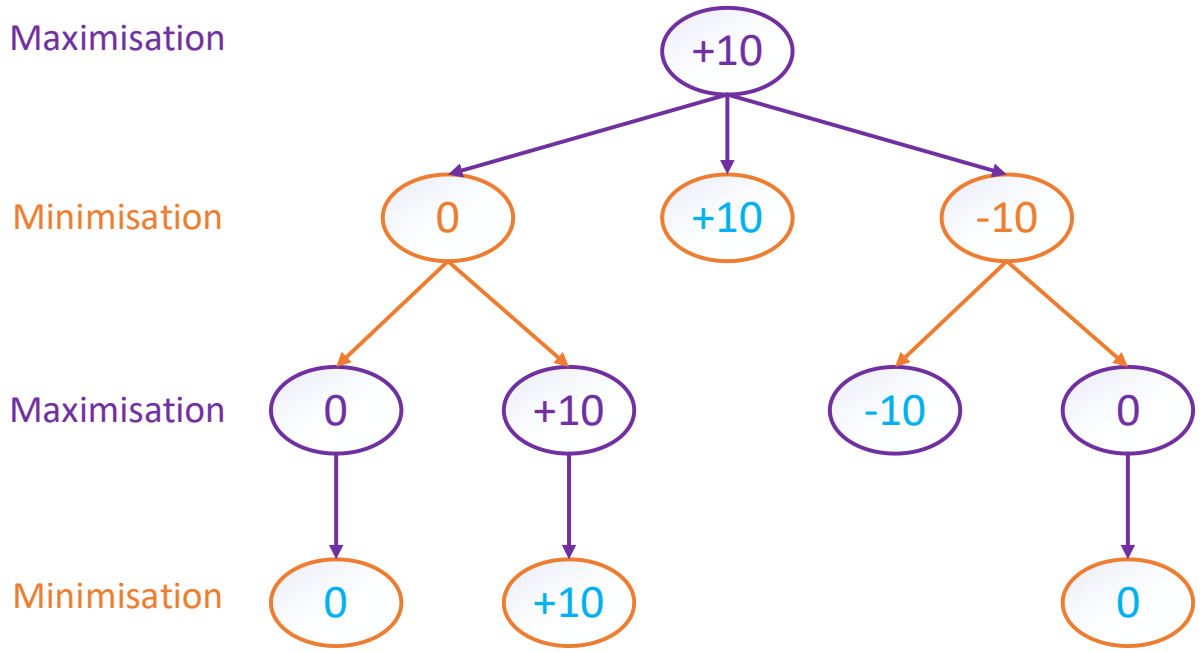


*Figure 7 : Exemple d'un schéma en arbre simplifié avec les attributions de scores par l'algorithme Minimax*

La dernière rangée, au tour de « X », minimise les résultats obtenus par la rangée d'en dessous. Mais comme dans cet exemple, il n'y a pas d'autre rangée, elle garde ses scores actuels.

La rangée d'au-dessus, au tour de « O », maximise les scores de la rangée d'en dessous. Cela signifie que le score gardé pour chaque disposition est celui du score le plus élevé obtenu à la rangée d'en dessous. Dans cet exemple, cette rangée ne possède qu'un seul score par possibilité, elle n'a donc pas de choix à faire, et garde les scores de la rangée précédente.

Sur la ligne d'en dessus, qui minimise, le score gardé parmi ceux de la rangée d'en dessous est celui dont la valeur est la plus faible. Sur la branche de gauche, « 0 » est moins élevé que « +10 », il sera donc conservé. Il en va de même sur la branche de droite, avec le score « -10 ».

Cela continue jusqu'à la racine (la première rangée), qui maximise. Le score le plus élevé sera donc le mouvement le plus optimisé. Dans cet exemple, « +10 » est le meilleur résultat, cela signifie que le meilleur coup à jouer consiste à placer la croix sur la ligne du milieu, à droite.

## 4.3 - Implémentation de l'algorithme dans le jeu

Afin de simplifier l'implémentation, l'algorithme a été divisé en trois fonctions. La première fonction a pour but de trouver le meilleur coup possible.

### *4.3.1 -Récupération du meilleur mouvement*

Pour cela, elle parcourt les cases vides du tableau, et ajoute son symbole (« X » si l'IA est le premier joueur, sinon « O ») dans la première case vide. Lorsqu'un symbole a été ajouté, elle appelle la fonction de minimisation. Cette fonction retourne une valeur, considérée ici comme un score. Si ce score est plus élevé que le précédent, elle le conserve (maximisation) ainsi que l'emplacement sur le plateau dans lequel le symbole a été ajouté. Ensuite, elle retire le symbole placé précédemment, et l'ajoute sur la prochaine case vide, et rappelle la fonction de minimisation, etc. Une fois le processus terminé, elle retourne les coordonnées X et Y de l'emplacement ayant obtenu le meilleur score.

### *4.3.2 -Fonction de minimisation*

Premièrement, elle contrôle si la partie est terminée. Si c'est le cas, elle renvoie un score, calculé comme vu précédemment :

- +10 si « X » gagne,
- 0 en cas d'égalité,
- -10 si « O » gagne.

Si la partie n'est pas terminée, elle parcourt le plateau à la recherche d'une case vide, et ajoute le symbole de l'opposant (ici « O ») dans la première case trouvée. Elle appelle ensuite la fonction de maximisation et récupère la valeur qu'elle renvoie. Si cette valeur est **moins élevée** que la précédente, elle la conserve. Puis, elle retire le symbole ajouté précédemment, l'ajoute dans la prochaine case vide, et rappelle la fonction de maximisation, et ce jusqu'à ce que toutes les cases vides aient été testées.

Une fois tous ces processus terminés, elle renvoie la valeur la plus faible qu'elle a obtenue.

### *4.3.3 -Fonction de maximisation*

Comme pour la fonction de minimisation, elle contrôle d'abord si la partie est terminée, et renvoie un score (déterminé comme pour la fonction de minimisation) si c'est le cas.

Si la partie n'est pas terminée, elle parcourt le plateau à la recherche d'une case vide, et ajoute son symbole (ici « X ») dans la première case trouvée. Elle appelle ensuite la fonction de minimisation et récupère la valeur qu'elle renvoie. Si cette valeur est **plus élevée** que la précédente, elle la conserve. Puis, elle retire son symbole ajouté précédemment, l'ajoute dans la prochaine case vide, et rappelle la fonction de minimisation, et ce jusqu'à ce que toutes les cases vides aient été testées.

Une fois tous ces processus terminés, elle renvoie la valeur la plus élevée qu'elle a obtenue.

## 4.4 - Résultats

Une fois ces trois fonctions implémentées et incorporées dans le jeu créé auparavant, il n'est pas possible de gagner une partie. En effet, comme l'IA choisit toujours les meilleurs mouvements, il n'est donc possible que de perdre ou de faire égalité.

### *4.4.1 -Exemple de partie où l'IA joue en premier*

Voici un résumé d'une partie lorsque l'IA est le premier joueur. Le nombre affiché en dessous représente le nombre de nœuds (dispositions) calculés par l'ordinateur afin de trouver le meilleur mouvement.

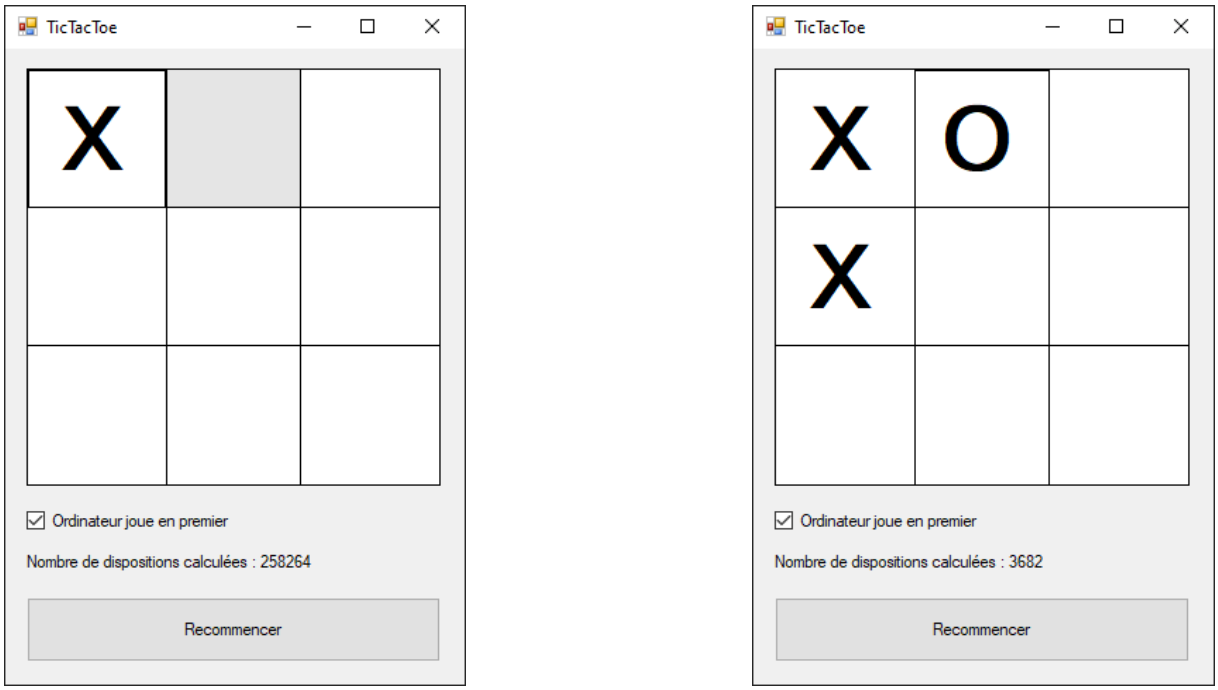


*Figure 8 : Démonstration d'une partie où l'IA joue en premier — Tour un à trois*

Le premier mouvement de l'IA sera toujours dans le coin en haut à gauche.

Le deuxième joueur place son symbole sur la case en haut au centre. L'IA répond en plaçant le sien sur la case au milieu à gauche. À partir de cette disposition, le deuxième joueur a de toute façon perdu :

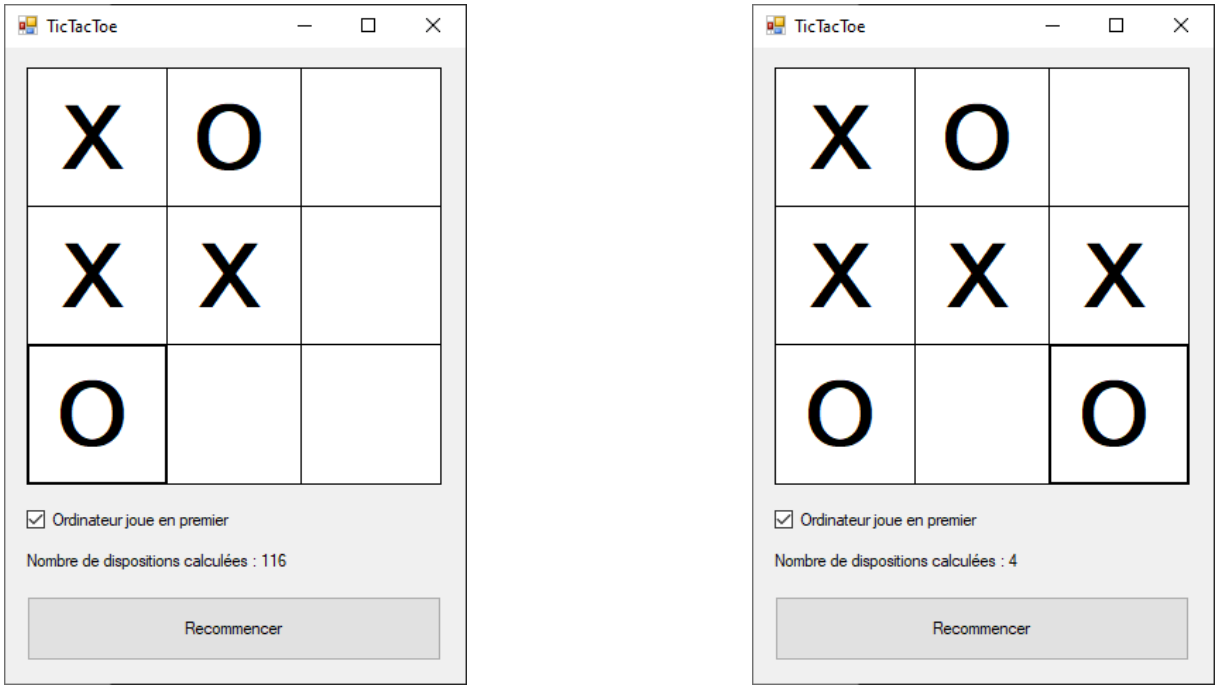


*Figure 9 : Démonstration d'une partie où l'IA joue en premier — Tour quatre à sept, victoire de l'IA*

En effet, il est obligé de placer son symbole sur la case en bas à gauche. Mais par la suite, l'IA place le sien au centre, créant ainsi deux lignes victorieuses : une en diagonale, et l'autre horizontale au centre. Peu importe le placement du deuxième joueur, celui-ci perd.

### *4.4.2 -Exemple de partie où l'IA joue en deuxième*

Et maintenant voici une partie lorsque l'IA est le deuxième joueur (l'IA garde son symbole « X ») :

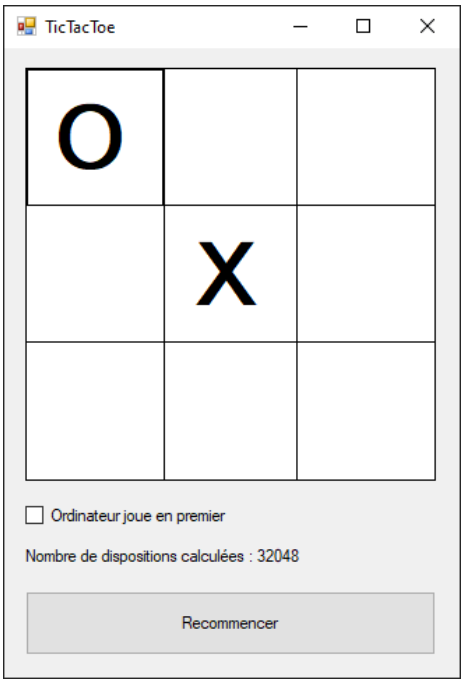


*Figure 10 : Démonstration d'une partie où l'IA joue en deuxième — Tour un à deux*

En imitant le premier coup de l'IA vu dans l'exemple précédent (coin en haut à gauche), celle-ci place son symbole au centre.

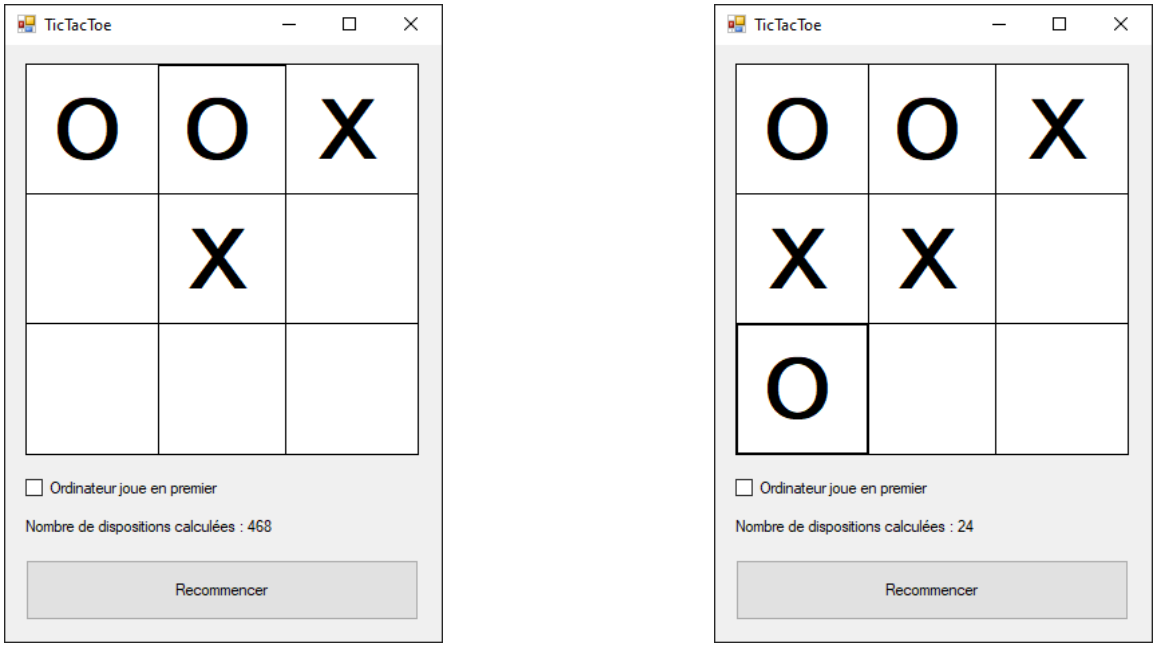


*Figure 11 : Démonstration d'une partie où l'IA joue en deuxième — Tour trois à six*

Le premier joueur continue sur la première ligne horizontale. L'IA le bloque et joue dans le coin supérieur droit. Pour ne pas perdre au prochain coup, le premier joueur doit jouer dans le coin inférieur gauche afin de bloquer la diagonale adverse. L'IA joue alors dans la case gauche afin de bloquer la ligne verticale gauche.

Le premier joueur se voit alors obligé de bloquer la ligne horizontale centrale, et termine ainsi

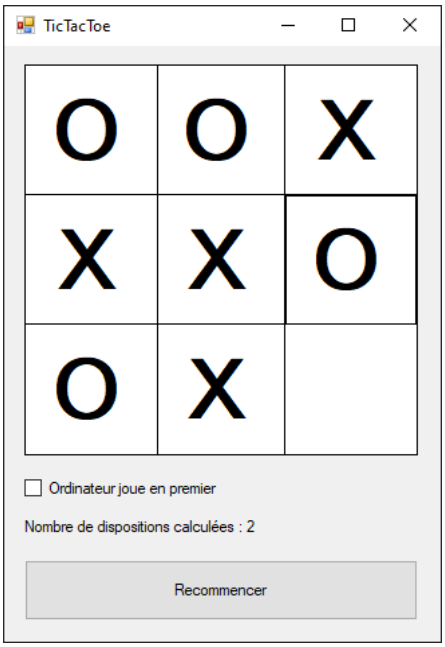


*Figure 12 : Démonstration d'une partie où l'IA joue en deuxième — Tour sept à huit, égalité*

la partie sur une égalité. En effet, les coups suivants n'importent pas, personne ne peut gagner à partir de cette disposition.

# 5 - Pour aller plus loin

## 5.1 - Simplification Negamax

Negamax est basé sur le même principe que Minimax, à la différence près que, au lieu de calculer la valeur optimale pour le joueur, il calcule la valeur la plus négative pour l'opposant. Ceci permet de trouver le moins bon coup que l'adversaire puisse jouer et ainsi de suite jusqu'à ce que l'IA gagne ou fasse égalité.

## 5.2 - Élagage alpha-bêta

Afin d'optimiser l'algorithme, il est possible de réduire le nombre de cas à calculer : c'est le rôle de l'élagage alpha-bêta.

Le principe est simple : intervenir dans l'algorithme de Minimax sans en changer le résultat. Pour ce faire, il faut évaluer les nœuds de l'arbre où les valeurs ne sont pas utiles.

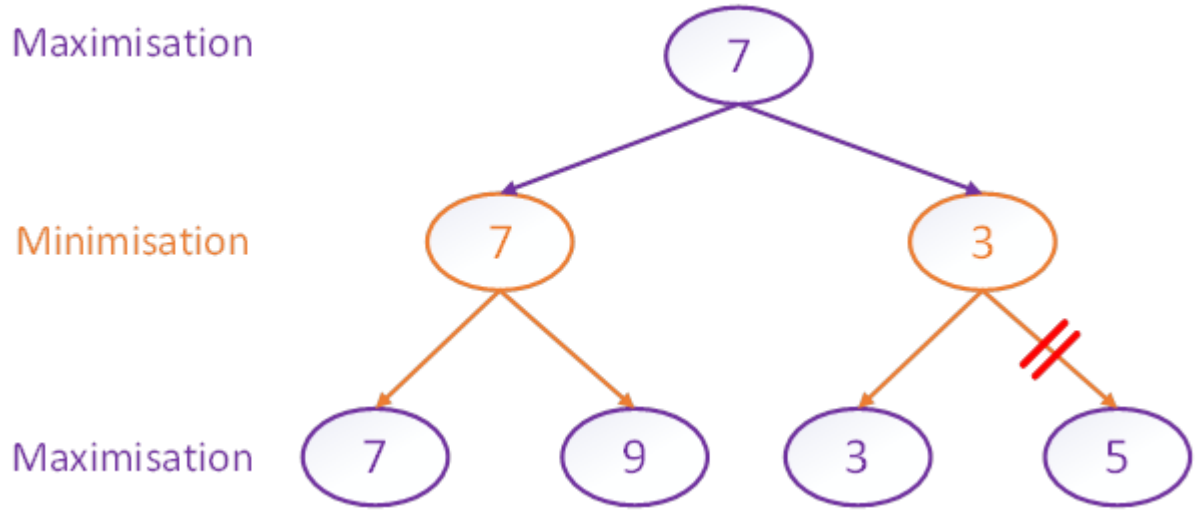


*Figure 13 : Exemple de schéma en arbre avec l'élagage alpha-bêta*

Sur ce schéma en arbre, la branche de gauche possède les valeurs 7 et 9 sur sa dernière rangée. Comme la ligne d'au-dessus minimise, elle choisira donc le 7.

Sur la branche de droite, la rangée centrale, qui minimise, choisira le 3 avant de calculer la valeur du nœud de droite (ici 5 sur le schéma). Mais comme elle minimise, elle ne changera de valeur seulement si celle-ci est inférieure à 3. C'est pourquoi il est alors inutile de calculer cette valeur : la première ligne maximise les résultats, elle prendra de toute façon la valeur 7.

7 est plus grand que 3, et le nœud avec la valeur 3 ne peut de diminuer son score.

## 5.3 - Machine learning

Le *machine learning*, ou apprentissage automatique en français, a pour but de permettre la réalisation de tâches non programmées, par une machine. Il est décomposé en deux phases distinctes.

La première phase comprend l'apprentissage de la machine. Élaborer un algorithme lui permettant d'évoluer dans un environnement inconnu. L'ordinateur est alors placé dans cet environnement, et au fur et à mesure des essais entrepris afin de réaliser la tâche demandée, il se formera alors un réseau neuronal, comparable au cerveau.

La tâche n'est pas explicitement demandée à la machine. Prenons l'exemple d'une voiture qui doit aller d'un bout à l'autre d'un parcours dans lequel elle se trouve, et ce sans toucher les murs qui l'entourent. La voiture est placée à l'entrée et elle essaie par elle-même d'arriver jusqu'à la fin du parcours sans lui donner d'autre instruction. Elle ne sait pas comment avancer, elle ne sait pas comment tourner, elle ne connaît pas les « règles » du jeu, mais elle est capable de le faire. C'est-à-dire qu'elle n'a pas conscience qu'elle peut tourner ou avancer, mais elle peut le faire.

Au bout d'un certain nombre de tentatives, la voiture sera capable d'aller jusqu'à la fin du parcours sans toucher un seul mur. Mais si elle est placée sur un parcours différent, elle devra alors recommencer le processus bien qu'elle sache déjà avancer. C'est ainsi que la première phase est réalisée.

La seconde est la mise en production de l'intelligence artificielle. Une fois que la machine a terminé son processus d'apprentissage, qu'elle s'est adaptée à plusieurs situations différentes et qu'elle est capable d'interagir correctement avec une nouvelle situation : elle peut alors être utilisée pour réaliser des tâches en production.

# 6 - Conclusion

Le jeu du morpion a été implémenté en C# et fonctionne parfaitement. Aucun problème n'a été rencontré lors de sa création.

Le chapitre compliqué de ce projet a été la conception d'un algorithme permettant de créer une intelligence artificielle capable de jouer au morpion, et ce sans jamais perdre une seule partie. Après quelques recherches, l'algorithme Minimax nous a semblé le plus efficace, tout en restant relativement simple à intégrer au projet.

Son implémentation ainsi que son association au jeu créé précédemment ont été réalisées sans contrainte ni problème majeur.

En définitive, la création d'une IA permettant de jouer au morpion est un succès. L'opposant ne peut que perdre ou faire une égalité.

Il aurait été possible d'aller plus loin en optimisant l'algorithme de Minimax grâce à Négamax ainsi qu'à l'élagage alpha-bêta. Mais comme le nombre de possibilités que l'ordinateur doit calculer est relativement faible, le processus est alors instantané ; il n'est donc pas nécessaire de l'optimiser.

# 7 - Bibliographies et références

### *7.1.1 -Sites Internet*

https://www.youtube.com/watch?v=trKjYdBASyQ&ab_channel=TheCodingTrain, Youtube, The Coding Train (chaîne youtube gérée par Daniel Shiffman), consulté le 02.12.2020. (Coding Challenge 154 : Tic Tac Toe AI with Minimax Algorithm [Vidéo]).

https://fr.wikipedia.org/wiki/Algorithme_minimax, Collectif Wikipedia, consulté le 02.12.2020 (Algorithme minimax)

https://fr.wikipedia.org/wiki/%C3%89lagage_alpha-b%C3%AAta, Collectif Wikipedia, Consulté 16.12.2020 (Élagage alpha-bêta)

https://en.wikipedia.org/wiki/Negamax, Collectif Wikipedia, consulté le 06.01.2021 (Negamax)

# 8 - Annexes

## 8.1 - Code source du morpion avec Minimax

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TicTacToe
{
    public partial class Form1 : Form
    {
        // Plateau de jeu
        string[,] board = new string[3, 3];
        const bool DefaultAITurn = true;
        bool xTurn = true;
        int nbTours = 0;
        Dictionary<string, int> scores = new Dictionary<string, int>();

        public Form1()
        {
            InitializeComponent();
            scores.Add("Personne", 0);
            scores.Add(«X», 10);
            scores.Add(«O», -10);

            InitialisationPlateau(null, null);
        }
        // Remettre le plateau à 0 (vider les cases)
        public void InitialisationPlateau(object sender, EventArgs e)
        {
            xTurn = DefaultAITurn;
            nbTours = 0;
            for (int i = 0; i < 3; i++)
            {
                for (int y = 0; y < 3; y++)
                {
                    Button btn = (Button)this.panel1.Controls["btn" + i +
"" + y];
                    board[i, y] = "";
                }
            }

            for (int i = 0; i < 3; i++)
            {
                for (int y = 0; y < 3; y++)
                {
                    Button btn = (Button)this.panel1.Controls["btn" + i
+""+ y];
                    btn.Text = «»;
                }
            }
```

```
            if (xTurn)
            {
                btnCases_Click(null, null);
            }
        }
        public string CheckWinner()
        {
            string gagnant = «»;

            // horizontale
            for (int i = 0; i < 3; i++)
            {
                if (board[i,0] == board[i,1] && board[i, 1] == board[i,2]
&& board[i, 0] != "")
                {
                    gagnant = board[i,0];
                }
            }

            // Verticale
            for (int i = 0; i < 3; i++)
            {
                if (board[0, i] == board[1, i] && board[1, i] == board[2,
i] && board[0, i] != "")
                {
                    gagnant = board[0,i];
                }
            }

            // Diagonales
            if (board[0, 0] == board[1, 1] && board[1, 1] == board[2, 2] &&
board[1,1]!= "")
            {
                gagnant = board[0,0];
            }
            if (board[2, 0] == board[1, 1] && board[1, 1] == board[0, 2] &&
board[2, 0] != "")
            {
                gagnant = board[2,0];
            }
            int openSpots = 0;
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    if (board[i, j] == "")
                    {
                        openSpots++;
                    }
                }
            }
            // S'il n'y pas de gagnant et que toutes les cases sont
remplies
            if (gagnant == «» && openSpots == 0)
            {
                return «Personne»;
            }
            else
            {
                return gagnant;
            }
```

```
        }

        private void btnCases_Click(object sender, EventArgs e)
        {
            Button btnSelectedCase = (Button)sender;
            if (xTurn)
            {
                string pos = MeilleurDeplacement(), coorX =
pos.Substring(0, 1), coorY = pos.Substring(1, 1);
                // MessageBox.Show(«X:" + coorX + "Y:" + coorY);
                Button btn = (Button)this.panel1.Controls["btn" + coorX +
"" + coorY];
                btn.Text = «X»;
                board[Convert.ToInt32(coorX), Convert.ToInt32(coorY)] =
"X";
                nbTours++; // Augmentation du nombre de tours
                xTurn = !xTurn; // Changement de joueur
                string gagnant = CheckWinner();
                if (gagnant != «»)
                {
                    MessageBox.Show(«Gagnant», gagnant + «a gagné !»);
                }
            }
            else
            {
                if (btnSelectedCase.Text == «»)
                {
                    int x =
Convert.ToInt32(btnSelectedCase.Name.Substring(3, 1));
                    int y =
Convert.ToInt32(btnSelectedCase.Name.Substring(4, 1));

                    btnSelectedCase.Text = «O»;
                    board[x, y] = "O";
                    nbTours++; // Augmentation du nombre de tours
                    xTurn = !xTurn; // Changement de joueur
                    // Contrôler s'il y a un gagnant
                    string gagnant = CheckWinner();
                    if (gagnant != «»)
                    {
                        MessageBox.Show(«Gagnant», gagnant + «a gagné !»);
                    }
                    else
                    {
                        btnCases_Click(null, null);
                    }
                }
            }
        }

        public string MeilleurDeplacement()
        {
            int bestMoveX=0, bestMoveY=0;
            int bestScore = int.MinValue+10;
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 3; j++)
                {
                    if (board[i,j] == "")
                    {
```

```
                    board[i,j] = "X";
                    int score = Minimisation(0);
                    board[i,j] = "";
                    if (score > bestScore)
                    {
                        bestScore = score;

                        bestMoveX = i;
                        bestMoveY = j;
                    }
                }
            }
        }

        return bestMoveX + "" + bestMoveY;
    }

    public int Maximisation(int prof)
    {
        string result = CheckWinner();
        if (result != "") return scores[result];
        int bestScore = int.MinValue+10;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (board[i, j] == "")
                {
                    board[i, j] = "X";
                    int score = Minimisation(prof+1);
                    board[i, j] = "";
                    if (score > bestScore)
                    {
                        bestScore = score;
                    }
                }
            }
        }
        return bestScore;
    }
    public int Minimisation(int prof)
    {
        string result = CheckWinner();
        if (result != "") {
                return scores[result];
        }
        int bestScore = int.MaxValue-10;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (board[i, j] == "")
                {
                    board[i, j] = "O";
                    int score = Maximisation(prof+1);
                    board[i, j] = "";

                    if (score < bestScore)
                    {
                        bestScore = score;
                    }
```

```
                    }
                }
            }
            return bestScore;
        }
    }
}
```