*École Polytechnique de l'Université de Tours*
*64, Avenue Jean*
*Portalis 37200*
*TOURS, France*
*(33)2-47-36-14-14*
www.polytech.univ-tours.fr

# Parcours des Écoles d'Ingénieurs Polytech

# Année 2022/ Année 2023

## Projet IA Puissance 4

**Étudiants**

**LOUET Noah**
noah.louet@etu.univ-tours.fr
**FOUGERAY Mattéo**
matteo.fougeray@etu.univ-tours.fr

**Encadrant**

**BILLAUT Jean-Charles**
jean-charles.billaut@univ-tours.fr

# Avertissement

Ce document a été rédigé par **Noah LOUET** et **Mattéo FOUGERAY** susnommés les auteurs. L'École Polytechnique de l'Université François Rabelais de Tours est représentée par **BILLAUT Jean-Charles** susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non-respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

# Table des matières

# Introduction

Le jeu du Puissance 4, également connu sous le nom de « Connect Four », est un jeu de stratégie pour deux joueurs qui consiste à placer ses jetons dans une grille verticale de 6 rangées et 7 colonnes. Le but étant d'essayer de former une ligne de quatre jetons horizontalement, verticalement ou en diagonale. Ce jeu simple est devenu un incontournable des soirées en famille et des allées de jeux depuis sa création dans les années 1970 par un cadre de la société américaine Milton Bradley. Depuis lors, le Puissance 4 est devenu un classique de la culture populaire, et est également largement joué en ligne sur des plateformes populaires comme Miniclip.

Notre projet, de développer une intelligence artificielle pour le Puissance 4 est né de notre expérience précédente en informatique. En terminale, nous avions créé une IA pour un autre jeu, le morpion. Or, pour le morpion, il était facile d'écrire à la main tout ce que devait faire l'IA. De plus, ce projet en terminale nous a poussé tous les deux à choisir des sujets de grands oraux en rapport avec l'IA notamment l'utilisation des arbres dans certains cas comme dans celui du minimax et a grandement contribué à notre intérêt porté pour cette face de l'informatique. C'est pourquoi cette année nous avons décidé de nous lancer dans un projet plus ambitieux : comment créer une IA pour puissance 4 ?

Dans ce rapport, nous présenterons les détails de notre projet d'IA de Puissance 4 qui a été entièrement réalisé en python. Nous verrons les différentes méthodes que nous avons utilisées pour développer cette IA et comment nous avons conçu le jeu et son interface graphique.

Ce rapport est organisé en plusieurs parties. Dans la première partie, nous expliquerons brièvement comment nous avons conçu les bases de l'interface graphique du jeu en utilisant la bibliothèque Tkinter de Python. Dans la deuxième partie, nous expliquerons comment nous avons mis en place le jeu de Puissance 4 en cohérence avec notre interface graphique. Dans la troisième partie, nous nous intéresserons aux différentes méthodes que nous avons abordées pour réaliser notre IA. Nous verrons tout d'abord notre essai d'IA grâce au machine learning. Ensuite, nous verrons comment et pourquoi nous avons finalement choisi la méthode du minimax au machine learning et comment nous l'avons développé pour notre IA. Nous discuterons également de la méthode de recherche alpha-bêta et de la manière dont nous l'avons utilisé pour optimiser notre IA avec aussi l'utilisation des threads.

Nous aborderons également tout au long de notre rapport les difficultés rencontrées pendant le développement du projet et les solutions que nous avons mis en place pour

les surmonter. Enfin, nous présenterons les résultats de notre projet, y compris les performances de notre IA et les potentielles améliorations que nous pourrions apporter à l'avenir.


Dans le cadre de notre projet d'IA pour un puissance 4, nous devons tout d'abord réaliser le jeu du Puissance 4. Pour l'interface graphique du jeu nous avons choisi d'utiliser la bibliothèque Tkinter en Python. De plus, nous avons choisi de séparer notre code en 2 fichiers distincts dans le but d'avoir plus de clarté. Un où nous allons réaliser toute l'interface graphique et le jeu. Puis dans le second fichier nous allons créer l'algorithme de l'IA dans le but de retourner la meilleure position à jouer en fonction d'une certaine grille. Ensuite, nous appellerons tout simplement cet algorithme dans notre fichier avec le Jeu afin de pouvoir jouer contre. Cette technique va notamment nous permettre de tester différents algorithmes et différents réglages de notre algorithme sans avoir besoin de réécrire tout le code dans notre fichier jeu. Nous n'aurons besoin que de l'appeler via le fichier externe. Mais avant tout ça, créons tout d'abord cette interface graphique.

## 1. Base de l'interface

### 1.1 Mise en place de la base

Comme nous l'avons dit, nous allons utiliser Tkinter. Tkinter est une bibliothèque de Python qui permet de créer des interfaces graphiques relativement facilement. Elle permet la création de nombreux widgets comme des boutons, des champs de saisie, des labels, etc. Tout ce qui est nécessaire afin de concevoir des interfaces graphiques interactives. Tkinter utilise le langage Tcl/TK pour créer l'interface graphique et permet de personnaliser l'apparence des widgets à l'aide de style.

Ici, nous utiliserons la classe Tk qui va permettre de générer la fenêtre principale de notre jeu. Nous importerons la bibliothèque Tkinter que l'on normera `tk` afin de n'écrire que ce préfixe devant toutes les utilisations du module pour permettre une meilleure visibilité du code.

### 1.2 Mise en route de la fenêtre

Maintenant que nous avons créé notre fenêtre, il nous fallait pouvoir la voir, afin d'y apporter toutes les modifications nécessaires. Nous avons tout simplement créé une fonction `main()` dans laquelle on initialise notre fenêtre grâce à `tk.Tk()`. Nous assignons cette fenêtre à la variable `root` qui va permettre grâce à `root.mainloop()` d'afficher notre fenêtre. Ensuite, pour que cette fonction s'exécute à chaque fois que nous lançons notre fichier, nous utiliserons une condition qui vérifie si le fichier est lancé, afin d'appeler notre fonction pour que celle-ci nous affiche notre fenêtre.

## 2. Création de notre Interface

### 2.1 Création de la Classe Jeu

Ici `Jeu` est tout simplement la classe que nous allons créer pour remplir notre fenêtre. En effet, nous avons choisi d'utiliser une approche orientée objet pour structurer notre programme. En POO (Programmation Orientée Objet), une classe, ici `Jeu`, est un modèle qui permet de définir la structure d'un objet. Un objet est une instance d'une classe et contient des attributs et des fonctions spécifiques. C'est pourquoi nous avons choisi d'utiliser une classe, car grâce à cela, nous pouvons créer notre objet afin de créer notre interface graphique. De plus, si nous avons envie de réutiliser notre interface graphique dans un autre projet, cela sera relativement simple. Nous aurons simplement besoin d'importer notre classe pour avoir notre jeu du Puissance 4.

Pour créer notre classe `Jeu`, il nous faut utiliser la fonction `__init__()`. La fonction `__init__()` est un constructeur, c'est elle qui va permettre d'initialiser la base de notre interface graphique et les attributs nécessaires à notre classe. Cette fonction nécessite d'un argument `self` qui est obligatoire. Il permet de désigner notre objet.

Ensuite, dans cette fonction, nous allons tout d'abord permettre la saisie d'un argument, qui sera la fenêtre mère. C'est la fenêtre qui va nous servir de base et dans laquelle nous allons créer notre jeu. Pour cela, nous allons créer une autre « Frame », une fenêtre, qui sera la fenêtre descendante de la fenêtre donnée en argument. Nous allons lui appliquer un padding (un écart) de 10 pixels sur les côtés afin que le contenu de notre jeu ne soit pas collé sur les côtés. Pour faire cela, nous utilisons la bibliothèque Tkinter et la fonction `Frame()`, à laquelle on va donner ses instructions. On va assigner cette fenêtre créée, à une variable qui se révélera être un attribut de notre objet afin d'y avoir accès à tout moment. Cela se forme comme `self.variable = valeur`, où self désigne notre objet comme dit précédemment.

Puis pour rendre cette modification effective, nous allons utiliser la méthode `pack()` sur la fenêtre. Ici une méthode et en fait une fonction propre à une classe qu'on utilise sur les objets.

Maintenant que nous avons notre base terminée, nous pouvons commencer à ajouter tous les éléments nécessaires à notre jeu. Durant tout le reste de ce chapitre, nous serons toujours dans la fonction `init()`, car c'est toujours l'initialisation de la fenêtre.

### 2.2 Mise en place du Titre

Nous commençons tout d'abord par l'ajout du Titre « Puissance 4 », pour cela, nous utilisons la fonction `Label()` de tkinter dans laquelle nous mettons le texte « Puissance 4 ». Cependant ici pour rendre effectif ce titre nous allons utiliser une autre méthode que `pack()` nous allons utiliser la méthode `grid()`. Cette méthode permet d'organiser de manière assez simple des éléments dans une grille. Ici, notre grille sera la fenêtre mère que nous avons créée. Donc avec cette méthode `grid()`, nous mettons effectif notre titre dans la première ligne, première colonne, tout en pensant à faire fusionner toutes les colonnes existantes afin d'avoir un titre centré grâce à l'attribut

`columnspan`. (Voir Annexe 1) Le prochain élément à créer maintenant et de loin le plus important est notre grille de Puissance 4.

### 2.3 Création de la Grille

Pour la création de notre grille, nous avons choisi d'utiliser un canvas, car cela permet une certaine optimisation en termes de coût mémoire. En effet, nous n'utiliserons qu'un seul et même élément pour toute notre grille, pas besoins de construire 42 boutons, mais seulement un canvas. Un canvas est communément une zone rectangulaire destinée à recevoir des dessins ou toutes autres figures. Ici, nous allons nous en servir pour dessiner notre grille. Pour cela, nous commençons par assigner à un attribut de notre objet, notre canvas avec comme argument un fond bleu pour que par la suite nous puissions créer des cercles blancs pour les emplacements de pion.

Pour créer ces emplacements, nous allons juste imbriquer 2 boucles l'une dans l'autre afin de créer 42 cercles blancs grâce à la méthode `create_oval()`. Pour savoir à quelles coordonnées créer ces cercles, nous allons justement utilisé ces deux boucles afin de connaître, en fonction de l'avancement de ces deux boucles, l'emplacement en x et y de notre cercle. (Voir Annexe 2)

### 2.4 Création du choix du mode jeu

Maintenant que nous avons notre grille de prête, nous allons créer un sélecteur de mode Jeu. En effet dans notre Puissance 4, nous voulions avoir la possibilité de pouvoir jouer au jeu de base, contre un autre joueur, mais aussi avoir la possibilité de joueur contre une IA. De plus, nous avons par la suite décidé de rajouter un mode IA vs IA afin de pouvoir faire jouer deux IA contre elles-mêmes et ainsi savoir laquelle est la meilleure.

Pour cela, nous allons créer une « Frame » que nous allons ajouter à côté de notre grille dans laquelle nous allons créer 3 radio bouton que nous alignerons tous à gauche grâce à la méthode `pack()`et l'argument `anchor=tk.W`. En effet, nous allons utiliser des radio bouton, car ici, seulement 1 seul mode ne pourra être joué à la fois. Et chaque Radio Button aura une valeur différente en fonction du mode sélectionné. C'est pourquoi il nous faut aussi créer une variable qui sera un attribut de notre objet `choix` qui sera modifiable en sélectionnant un des boutons afin de savoir quelle mode a été sélectionner. De plus, cette variable nous sera utile par la suite pour voir si un mode a été sélectionné ou non. (Voir Annexe 3)

### 2.5 Création du Bouton Jouer

Ensuite, il nous faut créer le bouton « Jouer » afin de lancer notre partie. Pour cela dans l'argument `command` du bouton, donc ce qui permet de dire quelle action faire quand on clique dessus, nous allons mettre une méthode nommée `jouer()` dont nous verrons le fonctionnement dans la deuxième partie de ce rapport qui va permettre de lancer la partie. (Voir Annexe 4).

### 2.6 Création de sélection de puissance

Pour finaliser la base de notre interface, nous avons rajouté la possibilité de modifier la puissance, c'est-à-dire donner la possibilité au joueur de jouer en puissance 2 à 6. C'est pourquoi, afin d'obtenir cette information, nous avons tout d'abord commencé par créer une variable `self.n`, qui sera un attribut de notre objet, initialisé à 4 qui nous sera utile tout au long de notre programme afin de savoir en quelle puissance nous sommes.

Ensuite pour modifier cette puissance nous avons rajouté un champ de texte qui sera possible de modifier grâce à un bouton placé en dessous relier à une méthode `udpaten()`

Cette méthode `updaten()` va tout simplement regarder la valeur du texte dans l'input, la convertir en int, donc en nombre afin de regarder si cette valeur se trouvait bien entre  2 et 6. Si c'est est le cas alors on modifie la valeur de `self.n`, ce qui permettra ensuite d'adapter toutes les règles du jeu et notamment comment notre IA devra réfléchir. De plus, cette fonction permet aussi de mettre à jour le titre de notre fenêtre en s'adaptant à la puissance pour que l'utilisateur puisse savoir avec quelle puissance il va jouer. (Voir Annexe 5)

Ici, nous pouvons déjà imaginer un premier problème avec la modification de la puissance au cours de la partie. C'est pourquoi nous avons créé une variable `play`, attribut de l'objet, initialisé à `False`, qui permet de savoir si une partie est en cours ou non. De plus pour une double sécurité, dans la méthode `jouer()`, nous avons fait en sorte de désactiver l'input lorsqu'une partie commence et de le réactiver lorsqu'une partie est finie, grâce à cela aucun risque de pouvoir changer la puissance durant une partie.

Ainsi, la base graphique de notre jeu a été réalisée, nous allons voir dans le deuxième chapitre comment rendre tout ça effectif en créant le Jeu du puissance 4. (Voir Annexe 6)

# 2 Création du Jeu

Dans la suite de notre rapport, nous allons présenter comment nous avons réalisé le jeu du Puissance 4. Pour ce faire, nous allons tout simplement compléter notre classe construire dans le chapitre 1 avec des méthodes afin de faire dérouler la partie.

## 1. Initialisation et lancement de la partie

### 1.1 Activation du bouton « Jouer »

Dans le premier chapitre, nous avons créé un bouton « Jouer », auquel nous avons attribué la méthode `jouer()` dans l'argument command. Tout d'abord nous allons vérifier qu'un mode de jeu a été sélectionner grâce à notre attribut choix créer plus haut et vérifier qu'il fait bien référence à un mode de jeu, sinon, auquel cas, on ne lance pas la partie.

Ensuite, vient l'initialisation de la partie. Cela commence par créer une nouvelle grille qui est une matrice de 6 lignes et 7 colonnes, les dimensions pour un puissance 4. Nous devons également réinitialiser la grille visuellement en redessinant les cercles blancs comme lors de l'initialisation.

Enfin, il faut lancer la partie, le but principal du bouton. Pour cela, on va créer une nouvelle variable, `Jturn`, attribut de notre objet qui sera là pour savoir à quel joueur il est de jouer. Nous l'initialisons aléatoirement entre la valeur 1 et 2, puis vient ensuite le premier coup à jouer par l'un des deux joueurs.

### 1.2 Activation de la grille

Ici, nous sommes toujours dans la méthode `jouer()` et c'est ici que notre attribut `choix` va être utile. En effet en fonction du mode de jeu choisi la partie ne se lance pas de la même manière.

Tout d'abord, le but ici va être de pouvoir faire jouer un joueur contre un autre. Pour cela, on va créer un attribut `play` qui sera là pour dire si un joueur IRL (pas l'IA) peut cliquer sur la grille.

Ensuite, dans les deux autres modes de jeu, pour jouer contre l'IA, nous vérifierons à chaque tour si un joueur est peut cliquer sur la grille en fonction de la valeur de `Jturn`. Ainsi, nous pourrons activer ou non la possibilité de cliquer sur la grille. Si le joueur ne peut pas cliquer sur la grille, c'est que c'est au tour de l'IA donc on appelle la méthode `IA()` pour qu'elle nous renvoie dans quelle colonne elle veut jouer en fonction d'une grille donnée.

Enfin pour le mode IA vs IA, dans ce cas, `play` reste à False et ne donne pas la possibilité au joueur de cliquer sur la grille.

## 2. Déroulement de la partie

### 2.1 Tour du joueur IRL

Pour ce qui est du tour du joueur, il nous fallait savoir comment détecter le clic du joueur sur le canvas et où placer le pion dans la grille. Pour cela, il existe une méthode pour les canvas : `bind()` qui permet d'associer un bouton de souris à une action lors du clic sur le canvas. On va donc associer cette action à une méthode que l'on appellera `callback()`. Cette fonction va, comme dans la méthode `jouer()`, vérifier l'état de la partie et la possibilité de joueur de jouer. De plus, nous pouvons récupérer les coordonnées x (l'abscisse) du clic de joueur, ce qui va nous permettre ainsi de savoir dans quelle colonne le jouer a cliqué. Pas besoins de vérifier les coordonnes y (l'ordonnée) vu que le pion doit être placé dans la ligne la plus en bas possible. Ainsi, cette méthode nous permet de savoir où le joueur veut jouer dans la grille.

### 2.2 Tour de l'IA

Pour la méthode `IA()`, le but est de récupérer un numéro de colonne valide afin de pouvoir la faire jouer dans la grille. Pour cela, nul besoin de vérifier l'état de la partie, car cette méthode est automatiquement appelée si nécessaire, avec les vérifications faites en amont. Pour déterminer en quelle colonne l'IA doit jouer, nous verrons ça grâce aux deux méthodes différentes que nous verrons dans les prochains chapitres. Le but sera de déterminer le meilleur coup à jouer et que ce soit un coup valide.

### 2.3 Traitement de l'information (mise à jour de la grille)

Pour ce qui est du traitement de l'information, cela se fera plus ou moins de la même manière dans les deux cas. Tout d'abord avec une vérification si le coup est valide ou non. Si le coup n'est pas valide, aucune action ne sera alors réalisée dans l'attente d'un coup valide.

Si le coup est valide, c'est-à-dire qu'il ne dépasse pas de la grille, nous allons tout d'abord l'ajouté dans notre grille virtuelle. Pour cela, on aura notamment besoin de créer une nouvelle méthode nommée `putgrille()`, qui permettra de déterminer la ligne du pion jouer en essayant de le placer le plus bas possible. Une fois cela fait, nous devons montrer à l'utilisateur que le coup a bien été pris en compte en modifiant notre canvas.

Ensuite, une fois que notre pièce est jouée et que nous avons les coordonnées x et y, nous pouvons alors les adapter en des coordonnées pour notre canvas et dessiné en fonction de la valeur de l'attribut `Jturn`, un cercle rouge ou jaune grâce à une méthode que nous avons créé, nommée `dessine()`.

### 2.4 Vérification de l'état de la partie

Maintenant que nous pouvons jouer, il faut donc créer les règles de ce Puissance 4 afin de pouvoir faire dérouler une partie et que deux joueurs puissent jouer l'un après l'autre. Pour cela, il nous faut vérifier l'état de la partie, savoir si c'est une égalité ou une partie gagnée. C'est pourquoi, en amont, lors de l'initialisation de la partie, nous avons créé une variable `gamesStatus`, attribut de l'objet qui permet de stocker l'état de la partie, initialisé à `True`, ce qui signifie que la partie est en cours.

Cependant, pour mettre à jour cette variable, il nous faut créer une méthode qui permet de vérifier si une partie est terminée. Cette méthode sera appelée après chaque coup joué et est nommée `gagne()`.

Pour cela, nous allons exécuter une boucle parcourant les index de 0 à 6, c'est-à-dire pour chaque colonne. À chaque fois nous allons regarder le dernier pion placé dans notre grille. Ensuite, pour ne pas faire de vérification inutile, nous vérifions si le pion est bien un pion de joueur qui vient de jouer. Sinon il est inutile de regarder, car ce n'est pas possible de faire gagner l'autre joueur directement en plaçant un pion.

La première vérification de victoire se fait tout d'abord horizontalement. Nous allons récupérer tous les pions à gauche et à droite du pion du même joueur et directement en contact. Ensuite, si leur nombre est supérieur à `self.n`, qui ici est la puissance choisie comme expliqué précédemment, alors la partie est gagnée pour le joueur et nous renvoyons `True`.

La seconde vérification se fait verticalement. Toujours en regardant ce pion, nous allons regarder tous les pions qui se trouvent en dessous et qui sont du même joueur et à la suite. Si leur nombre est supérieur ou égal à `self.n`, alors la partie est également gagné et nous renvoyons `True`.

La troisième vérification est la plus compliquée : il s'agit des diagonales. Pour cela, nous allons procéder en deux temps, une vérification pour la diagonale dite « négative », donc d'en bas à gauche vers en haut à droite, et une pour la diagonal « positive » donc d'en haut à gauche vers en bas à droite. Toujours à partir du pion que nous regardons, pour la diagonale négative, nous allons regarder tous les pions qui sont sur la même diagonale à sa gauche et qui se suivent et faire la même chose à droite. Si leur nombre et supérieur à `self.n`, alors la partie est gagnée et nous renvoyons `True`. C'est le même fonctionnement pour la diagonale positive.

Maintenant si après tout ça aucune victoire n'a été détecté alors `False` est renvoyé et un second test est effectué pour s'assurer qu'il existe encore une place dans la grille tout en haut. Si ce n'est pas le cas alors cette partie se termine sur une égalité.

Sinon, la partie continue en n'oubliant pas de changer la valeur de `Jturn` et, si nécessaire dans le cas du mode joueur vs IA, appeler la méthode `IA()` et de désactivé le clic sur la grille afin que la partie puisse continuer.

Ainsi le Jeu est fonctionnel tant du côté graphique qu'interne et permet de joueur au Puissance 4. (Voir Annexe 7)

Cependant, lors de nos parties de test, nous avons remarqué plusieurs problèmes. En effet, lorsque nous jouons, il était parfois difficile de déterminer où le dernier joueur avait placé son pion et quand bien même le jeu nous disait que la partie était finie, de trouver la victoire. C'est pourquoi nous allons voir dès maintenant comment nous avons pu résoudre ces problèmes.

# 3. Amélioration de l'aspect graphique

## 3.1 Affichage du dernier pion joué

Pour régler ce problème, nous avons eu l'idée de dessiné un cercle bleu clair autour du dernier pion joué, afin de pouvoir repérer rapidement où l'adversaire a joué. Cependant, pour réaliser cela, il nous fallait aussi stocker les coordonnées de ce pion afin de supprimer le cercle au prochain coup et ne plus le considéré comme le dernier pion joué. (Voir Annexe 8)

## 3.2 Affichage partie gagnée

Pour cela, nous avons choisi d'entourer les pions d'un cercle vert à la fin de la partie, de la même manière que le cercle bleu. Cependant, pour pouvoir encercler ces pions, nous avons dû stoker leurs coordonnées quelque part. C'est pourquoi notre méthode `gagne()`, en plus de renvoyé si la partie était gagnée ou non, renvoyait également les pions gagnants, stockés au fur et à mesure des vérifications. Ainsi, par la suite, il nous suffisait d'appeler notre méthode `dessinevict()` en donnant à chaque fois les coordonnées des pions stockés dans un tableau qui permettait de dessiner un cercle vert autour de la couleur initiale. (Voir Annexe 9)

## 3.3 Affichage égalité

Bien sûr, si la victoire était affichée, il nous fallait également afficher l'égalité. Pour cela, nous avons décidé de réaliser à un gradient de couleur à travers les 42 pions. Nous avons donc créé une nouvelle méthode `dessinega()` qui redessine au-dessus de chacun des 42 pions avec un gradient de couleur. Cela a été réalisé grâce à 2 boucles imbriquées qui ont permis de définir trois nombres compris en 0 et 255, qui nous permette de déterminer une couleur au format RGB. Cependant, il nous a fallu créer un convertisseur vers une couleur au format hexadécimal, car un canvas ne prend pas en compte le format RGB. Nous avons pu réaliser ceci grâce à la méthode `format()` en python qui permet de traduire une chaîne de caractère sous une autre forme suivant les arguments. (Voir Annexe 10)


# 1. Définition

## 1.1 Explication

Le machine learning, ou apprentissage automatique en français, est une branche de l'intelligence artificielle qui permet aux ordinateurs d'apprendre à partir de données et d'expériences passées pour effectuer des tâches sans avoir été explicitement programmés pour les réaliser. En d'autres termes, au lieu d'écrire des instructions précises pour que l'ordinateur effectue une tâche spécifique, comme identifier des images ou prédire des résultats, nous fournissons des exemples de données et des résultats correspondants pour entraîner un modèle d'apprentissage automatique. Le modèle apprend par lui-même à généraliser à de nouveaux exemples et peut être utilisé pour effectuer la tâche souhaitée sur de nouvelles données.

## 1.2 Exemples concrets

Par exemple, pour en revenir au domaine de la reconnaissance d'images, on peut entraîner un modèle de machine learning en lui fournissant des milliers d'images étiquetées (c'est-à-dire que l'on indique à l'ordinateur ce que représente chaque image). Le modèle utilise ensuite ces données pour apprendre à identifier les caractéristiques clé qui distinguent différentes catégories d'images (par exemple, la différence entre un chat et un chien), afin de pouvoir prédire correctement la catégorie d'une nouvelle image.

# 2. Lien avec notre projet

## 2.1 Avantages / Inconvénients

Le machine learning est une technologie qui présente à la fois des avantages et des inconvénients. L'un des avantages les plus importants est sa capacité à traiter de grandes quantités de données de manière rapide et précise. Cela permet de détecter des tendances et des modèles qui seraient difficilement perceptibles par les humains, et d'automatiser des tâches répétitives et fastidieuses. De plus, ce modèle est très adaptatif, en effet, il se base sur les données entrées en paramètres, si celles-ci évoluent, la perception de ce modèle évolue aussi en réponse. Cependant, le machine learning présente également des inconvénients. Il peut être très coûteux et chronophage de collecter des données de haute qualité nécessaire pour entraîner les modèles de machine learning, et les modèles peuvent être biaisés s'ils sont entraînés sur des données non-représentatives ou déséquilibrée. De plus, les modèles de

machine learning peuvent être très complexes et difficiles à comprendre, ce qui peut rendre compliquée l'explication de leurs décisions.

### 2.2 Lien concret

Dans le cas du jeu de Puissance 4, le machine learning peut être utilisé pour créer une intelligence artificielle capable de jouer au jeu à un niveau élevé. Entraîner un modèle de machine learning à jouer à Puissance 4 implique de lui fournir de nombreuses parties jouées par des humains ou d'autres modèles d'IA, afin qu'il puisse apprendre à identifier les stratégies gagnantes et à anticiper les coups de son adversaire.

Une fois que le modèle a été entraîné, il peut être utilisé pour jouer au jeu de manière autonome, en choisissant les coups les plus susceptibles de mener à la victoire. L'IA peut également être programmée pour s'adapter à différentes situations de jeu, en utilisant des algorithmes de prise de décision pour choisir la meilleure stratégie possible. Le machine learning peut donc jouer un rôle important dans le développement d'une IA pour le jeu de Puissance 4.

## 3. Implémentation

### 3.1 Théorie

L'implémentation d'un programme de machine learning pour jouer au jeu de Puissance 4 implique plusieurs étapes clé. Tout d'abord, il faut collecter des données d'entraînement sous forme de parties jouées par des humains ou d'autres modèles d'IA. Ces données peuvent aussi être générées grâce à des parties virtuelles où le programme joue et apprend par lui-même. Ces données sont ensuite utilisées pour entraîner un modèle de machine learning à reconnaître les stratégies gagnantes et à anticiper les mouvements de l'adversaire.

Une fois que le modèle a été entraîné, il est nécessaire de l'évaluer en utilisant des données de test pour vérifier s'il peut prédire correctement les mouvements de l'adversaire et choisir les coups les plus efficaces. Si le modèle ne performe pas de manière satisfaisante, il peut être nécessaire de le retravailler ou de collecter davantage de données d'entraînement pour améliorer ses performances.

Ensuite, le modèle peut être utilisé pour jouer au jeu de manière autonome, en choisissant les coups les plus susceptibles de mener à la victoire en fonctions des mouvements de l'adversaire. Il est important de noter que l'implémentation d'un programme de machine learning pour jouer à Puissance 4 peut être très complexe, car le jeu implique de nombreuses variables et stratégies, et le modèle doit être capable de prendre des décisions en temps réel.

Enfin, l'implémentation d'un tel programme pour jouer au Puissance 4 peut être long et complexe, mais une fois achevé, il peut se révéler être efficace et peut offrir une expérience de jeu stimulante et pertinente.

### 3.2 Mise en pratique

Pour réaliser cet algorithme, nous avons décidé d'utiliser une base de données grâce à la bibliothèque `sqlite3` dans laquelle nous allons stocker les grilles sous forme de chaîne de caractère avec le coup joué en y associant le potentiel de victoire avec ce coup sur cette grille.

Pour se faire avant d'entrer des données dans la BD, il nous fallait simuler le jeu. Pour cela, on a repris notre fonction qui testait l'état d'une partie afin de détecter une victoire ou une égalité.

Durant cette partie, on construit 2 types de grille, une sous la forme de matrice pour tester l'état du jeu et une autre sous forme de chaîne de caractère avec l'ordre des colonnes dans lesquelles les pions ont été joués afin de pouvoir le stocker dans la base de données et de pouvoir utilisé ces données en étant le joueur 1 ou 2.

Ensuite, afin de simuler la partie, nous allons tour par tout appeler notre fonction `select_action()` qui va retourner l'indice dans la colonne jouer en fonction d'une grille donnée. Pour se faire va cette fonction aura 2 choix, soit joué un coup au hasard afin de découvrir un potentiel bon placement ou essayé de jouer en fonction des données stockées. Ce choix se fait sur une probabilité que l'on donnera en argument en fonction de nos préférences. Par exemple au début de son entraînement, il est plus efficace qu'elle recherche plein de différents coups pour avoir une certaine base donnée et par la suite essayer de s'entraîner avec.

Maintenant afin de stocker tous ces coups jouer nous allons une fois la partie finie attribué une récompense. Pour se faire pour tous les coups qui ont mené à la victoire, nous allons soit les ajouter s'ils ne sont pas dans la BD avec une évaluation de 1. Sinon, s'ils sont déjà présents dans la BD alors on va prendre l'évaluation déjà présente à laquelle on ajoute 1 que nous allons diviser par le nombre de fois où ce coup a déjà été joué. Cependant pour les coups perdants, alors on effectue le même schéma de pensée, mais cette fois-ci avec une récompense qui correspond à -1.

Ainsi, grâce à cela au fil du temps, on obtient une base de données composée de différentes grilles avec un coup attribué, une probabilité qui correspond au potentiel de victoire et le nombre de fois que ce coup a été joué. Ceci va nous permettre d'en déduire le meilleur coup à jouer en fonction d'une grille par la suite.

Par exemple, après avoir simulé 3000 parties à la suite, l'algorithme a déduit de lui-même que le meilleur coup à jouer en premier était au milieu (Voir Annexe 11), ce qui est le cas. Donc nous pouvons voir que ceci fonctionne, mais que ça peut prendre beaucoup de temps avant d'avoir un algorithme vraiment performant.

### 3.3 Décision finale

Finalement, nous avons décidé de ne pas retenir cette solution et de choisir un algorithme de type « minimax », et ce, pour plusieurs raisons.

Le choix de l'algorithme Minimax plutôt que le machine learning pour le jeu de Puissance 4 présente plusieurs avantages. Tout d'abord, l'algorithme Minimax offre une grande précision et un contrôle total sur le fonctionnement de l'IA. Contrairement aux modèles de machine learning, qui peuvent être difficiles à comprendre et à contrôler, l'algorithme Minimax est basé sur des règles claires et peut être ajusté et optimisé selon les besoins. Cela permet de mieux comprendre les résultats de l'IA et de les adapter plus facilement en fonction des besoins.

En outre, l'algorithme Minimax ne nécessite pas de grandes quantités de données d'entraînement pour fonctionner efficacement. Il est basé sur des règles de jeu claires et peut être entraîné à partir de quelques exemples de parties, ce qui peut être beaucoup plus facile et moins coûteux que la collecte de données massives nécessaires pour entraîner des modèles de machine learning. De plus, l'algorithme Minimax est conçu pour être rapide et efficace, ce qui le rend idéal pour les jeux en temps réel comme le Puissance 4. Contrairement aux modèles de machine learning, qui peuvent prendre beaucoup de temps pour traiter les données et prendre des décisions, l'algorithme Minimax peut prendre des décisions rapidement en utilisant des règles simples et des calculs rapides.

Un autre avantage de l'algorithme Minimax est qu'il est basé sur des stratégies claires, ce qui le rend plus facile à comprendre et à analyser que les modèles de machine learning, qui peuvent être opaques et difficiles à interpréter. Cela permet de mieux comprendre les résultats de l'IA et de les adapter plus facilement en fonction des besoins. Enfin, l'algorithme Minimax est moins sujet aux biais que les modèles de machine learning, qui peuvent être sujets à des erreurs si les données d'entraînement ne sont pas représentatives ou si le modèle n'est pas correctement ajusté.

En résumé, l'algorithme Minimax offre une alternative plus précise, contrôlable, rapide et moins sujette aux biais que les modèles de machine learning pour le jeu de Puissance 4. Cela permet de créer une IA plus rapide, plus efficace et plus facile à comprendre pour les joueurs.

# 4 Algorithme Minimax

Dans cette partie, nous allons présenter le principe d'un algorithme Minimax, qui est la méthode que nous avons finalement choisi d'utiliser pour notre intelligence artificielle. Nous commencerons par expliquer la partie théorique de cet algorithme, qui consiste à simuler un certain nombre de coups afin d'en déduire le meilleur coup à jouer. Ensuite, nous expliquerons comment nous avons implémenté cet algorithme afin qu'il puisse répondre à nos besoins sur le puissance 4 et qu'il puisse s'adapter en fonction des différentes puissances. Enfin, nous verrons comment nous avons optimisé notre algorithme à travers l'utilisation de techniques d'élagage alpha-bêta et de parallélisation des threads, dans le but d'optimiser le temps d'exécution de notre programme.

## 1. Théorie

### 1.1 Explication de la notion d'arbres

Tout d'abord, avant d'aborder l'explication de l'algorithme, nous allons aborder une notion qui est importante à connaître qui est celle des « arbres ». Ici, un arbre est une structure de données qui va notamment permettre de représenter les différentes options lors d'un processus de décision. Dans notre cas, cela va nous permettre de représenter les différentes options de grille possibles suite à un coup joué par un des joueurs.

Dans un arbre, il y a ce qui s'appelle les « nœuds » qui représentent un état du problème initial. Entre ces différents nœuds, il y a les « branches » qui représentent l'action effectuée afin d'arriver à ce sous-problème. C'est-à-dire qu'ici à la racine de notre arbre, nous aurons notre grille initiale, la grille qui représente l'état du jeu, puis nous allons ensuite simuler toutes les grilles possibles qui découlent de celle-ci jusqu'à une certaine profondeur. Une fois cette profondeur atteinte, nous arrivons aux feuilles de l'arbre qui va représenter l'une des possibilités les plus avancée du problème initial auquel nous allons affecter une fonction d'évaluation (dont nous allons le voir le détail plus tard) afin de déterminer la valeur de la grille.

(Voir Annexe 12)

### 1.2 Explication fonctionnement et but de l'algorithme Minimax

L'algorithme minimax est un algorithme qui va nous permettre de déduire le meilleur coup possible à jouer en fonction d'une grille donnée. En effet, cet algorithme a pour but de déterminer le meilleur coup possible en considérant les coups adverses, et

imagine toujours que l'adversaire jouera le coup qui nous mettra dans la pire situation possible.

Pour réaliser ceci, cet algorithme se repose sur la notion des arbres. En effet, il va construire un arbre qui va représenter toutes les grilles possibles jusqu'à une certaine profondeur, c'est-à-dire jusqu'à un nombre maximal de coup jouer.

Ensuite, afin de déterminer le meilleur coup à jouer, il va attribuer une valeur grâce à une fonction d'évaluation sur les grilles des feuilles de l'arbre. Puis il va comparer les valeurs des sous-nœuds qui descendent d'un même nœud, en choisissant la valeur maximale quand c'est à son tour de jouer et la valeur minimale quand c'est à l'adversaire. Cela se fait en alternance, en remontant l'arbre jusqu'à la racine qui représente le coup à jouer pour la grille initiale.

Ainsi, en faisant ça il permet de déterminer le meilleur coup possible à jouer en fonction des coups possible dans le futur tout en imaginant que l'adversaire jouera de manière optimale. (Voir Annexe 13)

### 1.3 Pourquoi le choix de l'algorithme Minimax

Tout d'abord, comme nous avons pu le voir, cet algorithme s'adapte parfaitement à nos besoins pour le puissance 4.

En effet, notre but ici étant de déterminer le meilleur coup possible afin de gagner tout en bloquant notre adversaire de gagner, cet algorithme répond parfaitement à cette demande.

De plus comparé au machine learning cet algorithme n'a pas besoin de s'entraîner et peut être effectif à tout moment, sa performance dépend seulement de la fonction d'évaluation.

## 2. Implémentation

Maintenant que nous avons vu le côté théorique, il est temps de l'implémenter, pour se faire comment dit précédemment nous allons créer cet algorithme dans un fichier tiers. Pour se faire nous avons créé une fonction `algon_th()` qui désigne notre intelligence artificielle capable de s'adapter en fonction des différentes puissances de n.

Pour se faire nous avons 4 arguments, `grilla` qui représente la grille de la partie, remplit de 0,1 et 2, `nb` afin de savoir quel joueur nous sommes dans la grille, `pui` afin de savoir selon quelle puissance nous devons réfléchir et enfin `depthinit` qui représente la profondeur de recherche que nous voulons.

Nous allons voir la création de cette fonction en 2 parties, une pour la création de l'arbre et l'autre pour la méthode d'évaluation des feuilles.

### 2.1 Mise en place de l'arbre

Tout d'abord commet dit plutôt, il nous faut créer notre arbre de simulation. Pour cela, nous allons créer la fonction `minimax()` qui va prendre en argument la grille courante, la profondeur qu'il reste à faire, `nb` qui représente notre numéro, `Max` qui sera un

booléen afin de savoir s'il faut chercher la valeur maximale ou minimale en remontant notre arbre puis 2 paramètre appelé alpha et beta dont nous verrons l'utilité plus tard avec l'élagage alpha-bêta.

Cette fonction est en fait une fonction récursive, car on va chercher à répéter la même action tout à long de notre arbre et ensuite pouvoir remonter les différentes valeurs des nœuds.

Tout d'abord, pour lancer nos simulations, il y a une petite phase d'initialisation. Pour ce faire, nous allons créer un tableau de longueur 7, afin de stocker l'évaluation des coups possible dans la grille en fonction de la colonne. Ensuite, nous regardons tout d'abord s'il est possible de jouer dans la colonne, sinon on affecte juste la valeur -infini à cette colonne ce qui signifie qu'il est impossible de jouer.

Si c'est possible de jouer, nous allons donc appeler notre fonction `minimax()` avec en argument la grille avec un pion rajouté en fonction de l'index de notre boucle grâce à notre fonction `putgrille()` qui va chercher à mettre un pion de joueur le plus bas possible dans une grille en fonction de la colonne donnée. C'est comme ça que nous créons nos 7 premières simulations de grille. Ensuite pour l'argument `Max` on donne False pour qu'on recherche le minimum, car là, c'est l'adversaire qui va jouer, donc dans la fonction `minimax()`, nous allons chercher les grilles avec le score le plus bas, c'est-à-dire les meilleurs coups pour notre adversaire.

Après avoir lancé nos 7 premières simulations, c'est au tour de la fonction `minimax()`, c'est elle qui va prendre en charge la création de l'arbre de simulation jusqu'aux feuilles et la remonter de leur évaluation comme expliqué plutôt.

Pour cela dans cette fonction, nous regardons déjà si la grille donnée en paramètre n'est pas une grille d'une partie terminée, c'est-à-dire soit l'un des deux joueurs à gagner ou qu'il y a égalité, auquel cas on s'arrête puis on renvoie une valeur déterminée par la fonction d'évaluation. Il y a aussi une deuxième condition d'arrêt, qui est la profondeur maximale, donc ici quand on atteint la profondeur 1, c'est-à-dire qu'on se trouve au niveau des feuilles de notre arbre, on arrête les simulations puis on renvoie la valeur de la grille déterminée par la fonction d'évaluation.

Ensuite si aucun des conditions de dessus n'est rempli, c'est qu'il faut continuer de construire l'arbre des simulations. Pour cela, nous allons faire de la même manière que dans l'initialisation, tout en n'oubliant pas de diminuer la profondeur de 1 et de changer la valeur de Max afin d'intervertir entre le choix du minimum et du maximum expliqué plutôt.

Maintenant que nous savons comment créer notre arbre, on va s'intéresser à comment remonter les valeurs retournées au bout des feuilles. Pour cela, lorsque l'on va chercher le maximum, on va juste rechercher la valeur maximum retournée par les sous-nœuds et ensuite retournée cette valeur au nœud précédent, et la valeur minimum dans le cas où le but serait de chercher la grille qui nous met dans la pire position.

Ainsi, après que tout l'arbre soit résolu et que toutes les valeurs soient remontées jusqu'aux 7 simulations de départ, on obtient un tableau avec 7 valeurs. Il ne nous reste plus qu'à renvoyer l'index de la valeur maximale du tableau afin de savoir en quelle colonne il faut jouer.

## 3. Méthode d'évaluation

Maintenant que nous avons un arbre fonctionnel, il nous faut créer notre méthode d'évaluation. Cette méthode est la partie la plus importante de cet algorithme, c'est ce qui va justement vraiment faire la différence entre 2 algorithmes minimax.

Pour rappel, cette méthode va permettre d'affecter aux feuilles une valeur en fonction de si oui ou non arriver à cette grille est bon ou non pour nous.

### 3.1 Partie terminée

Nous avons décidé de considérer les nœuds où les grilles étaient des partis terminés comme des feuilles afin d'avoir un gain de temps. En effet, il est inutile de continuer de simuler des coups sur une grille où la partie est déjà terminée.

Lorsque que nous rencontrons une victoire, nous allons renvoyer une valeur plus haut que la normal, par exemple 1000, que nous allons multiplier en fonction de la profondeur qu'ils nous restent afin de gagner le plus vite possible. Ensuite, nous allons multiplier cette valeur par -1 si `MAX` vaut `True`, car cela signifie que c'est l'adversaire qui a gagné dans le cas contraire, on multiplie par 1 car c'est nous qui avons gagné.

S'ajoute à ça la valeur de la grille sans les victoires, renvoyée par la fonction d'évaluation que nous allons voir plus tard afin d'obtenir plus de précision quand nous avons par exemple 2 parties gagnées à la même profondeur au cas où l'adversaire arriverait à nous contrer. Dans le but de nous permettre de nous diriger vers une grille qui même si nous ne gagnons pas, nous dirige vers la grille avec le plus de chance de gagner. Nous pouvons voir ça comme des probabilités de gagner, nous allons nous diriger vers la probabilité la plus haute même si ce n'est pas 100 %.

Maintenant si nous rencontrons une égalité alors cela ne signifie pas que c'est une défaite, ni une victoire, donc nous allons renvoyer 0, une valeur neutre. Ce qui va donc pousser notre IA à plutôt aller chercher une égalité quand cela est possible plutôt qu'une défaite lorsqu'il n'y a plus aucun moyen de gagner.

### 3.2 Fonction d'évaluation

Nous allons maintenant passer à la partie la plus compliquée de cet algorithme, celle de la création de la fonction d'évaluation. Cette fonction s'est révélée d'autant plus compliquée, car elle devait pouvoir s'adapter aux différentes puissances.

Pour cette fonction d'évaluation, nous avons décidé de ne pas compter les pions qui formaient des victoires, car ceux-ci étaient déjà comptés lorsqu'une partie était gagnée. Notre objectif ici était de détecter les potentielles possibilités de réaliser des

alignements de puissance en fonction de la position de nos pions, mais aussi de ceux adverse.

Pour cela, il fallait aussi réfléchir aussi comment optimiser cette fonction pour qu'elle ne soit pas trop chronophage, le but étant que notre algorithme puisse jouer un minimum rapidement. C'est pourquoi dans un premier temps nous avons décidé comme dans la fonction qui détecte les victoires de ne regarder seulement les pions qui se trouvaient au plus haut sur chaque colonne. En effet, les pions déjà entourés ou bloqués n'ont que très peu d'intérêt car il n'y a souvent pas possibilité de réaliser des puissances. De plus, si ces pions servent à réaliser une puissance alors ils seront détectés grâce aux pions en surface qui seront en lien avec.

Puis pour avoir une évaluation plus précise nous allons appliquer cette fonction sur nos pions qui vont donc rajouter des points à l'évaluation et sur les pions adverses qui eux vont aux contraires les diminuer.

Cette fonction d'évaluation va avoir pour but de retourner un tableau de longueur puissance -2, car on ne compte pas les alignements de 1 pion ni ceux de la puissance car ils sont déjà comptés. Ensuite à chaque index i, on va avoir le nombre d'alignement de i + 1 pions qui peuvent potentiellement être utile pour faire une puissance. Dans ces alignements, on compte ceux horizontales, verticales et en diagonales tous confondus.

Pour faire cela pour chaque pion en surface, on va regarder leur potentiel en horizontal, vertical et en diagonale.

Pour l'horizontale, on va tout simplement regarder toute la ligne, puis la découper en différents morceaux de la longueur de la puissance. Ensuite, s'il y a d'autres pions que celui initiale assez proche pour plus tard faire une puissance sans qu'il n'y ait bien sûr de pion adverse entre eux, alors on compte le nombre de pions, même s'ils ne sont pas directement à côté, comme un alignement de n pions. On le compte, car même s'ils ne sont pas à côtés, ils peuvent quand même représenter une certaine menace. Ensuite nous faisons ça pour chaque pion en faisant bien attention de ne pas regarder deux fois la même ligne.

Pour la verticale, là c'est beaucoup plus simple, nous regardons toujours pour chaque pion en surface le nombre de pions aligner de la même couleur en contact en dessous, ce qui forme n pions. Ensuite on regarde si au-dessus il y a encore assez de place pour former une puissance verticale en prenant en compte déjà ceux aligner, si c'est le cas, alors on compte cet alignement.

Pour la diagonale, cela va fonctionner de la même manière que l'horizontale, c'est-à-dire qu'on regarde toute la diagonale et en réalisant le même cheminement de pensée.

Ainsi, à la fin de tout ça, nous obtenons 2 tableaux avec certaines valeurs qui correspondent aux nombres d'alignement potentiel, un pour nous et l'autre pour l'adversaire. Ensuite, pour en sortir une évaluation, nous ajoutons en fonction du nombre d'alignement et de leur longueur une certaine valeur, qui est de plus en plus

importante en fonction de longueur. Puis il ne nous reste plus qu'à renvoyer cette valeur pour qu'elle puisse être remontée à travers l'arbre.

## 3. Optimisation

Nous allons maintenant parler de comment nous avons optimisé cet algorithme. En effet, bien qu'il soit très puissant, il peut s'avérer très chronophage si nous l'utilisons de manière assez brutale. C'est pourquoi nous avions déjà délimité la durée d'exécution en fixant une certaine profondeur maximale, car il serait impossible de générer absolument toutes les grilles dès le premier coup. C'est d'ailleurs pour ça qu'il existe la fonction d'évaluation, afin de « prévoir » ces prochaines grilles. Cependant, nous avons utilisé deux autres techniques qui permettent que notre algorithme soit beaucoup moins chronophage.

### 3.1 Threads

En python, l'exécution d'un programme se fait de manière linéaire. C'est-à-dire que chaque instruction va s'exécuter l'une après l'autre. En temps normal cela ne pose pas de problème, mais pour la génération de notre arbre cela peut s'avérer être une grosse perte de temps. En effet, si l'on pouvait lancer les 7 sous-arbres à partir des 7 premières simulations en même temps, cela serait un gain de temps plutôt conséquent. C'est pourquoi nous avons décidé d'implémenter la parallélisation threads dans notre programme.

Pour être un peu plus précis, notre programme s'exécute normalement sur un seul thread (processus). L'idée ici va être de lancer sept threads au lieu d'un afin d'avoir un temps d'exécution beaucoup plus rapide. Pour cela, nous avons utilisé la libraire `ThreadPool` qui permet de créer différents threads dans un programme.

Nous allons l'utiliser lors de la création des simulations de base. Puis afin de comparer les différentes valeurs du tableau qui correspondent aux colonnes, nous allons utiliser la méthode `get()` qui permet d'obtenir la valeur retournée par la fonction mais aussi d'attendre que celle-ci soit retournée avant de continuer la suite du programme. C'est pourquoi il faut bien lancer ses différents threads avant de demander la valeur retournée sinon les threads ne serviront à rien.

Ainsi, grâce à cette méthode, nous avons pu observer une amélioration du temps d'exécution considérable, d'environ 5 secondes, surtout pour les premiers coups qui sont les plus longs à trouver. Cela représente en profondeur 8 à peu près un quart du temps, avant l'utilisation des threads. On a pu notamment voir cette différence grâce au mode IA vs IA en utilisant un programme avec les threads (2) et un sans les threads (1) .(Voir Annexe 14)

### 3.2 Élagage Alpha-bêta

L'élagage Alpha-bêta est une technique d'optimisation qui permet de réduire le nombre de nœuds exploré dans un arbres en supprimant des branches qui ne sont pas intéressantes de regarder.

Comme nous l'avons vu plutôt nous allons utiliser deux variables alpha et beta. Alpha représente la valeur maximale et beta la valeur minimale que le joueur en cours peut obtenir. C'est pourquoi au départ alpha est initialisé à -infini et beta à +infini afin que l'on puisse modifier ces valeurs en fonctions des nœuds visités.

Ce changement va s'effectuer après l'évaluation d'un nœud. Lorsque que l'on cherche le maximum, on va regarder si l'évaluation est supérieure à alpha, alors on assigne à alpha l'évaluation, et si en plus l'évaluation est supérieure à beta, alors on retourne la valeur de beta, car cela ne sert à rien de regarder les nœuds à côté, car le joueur a déjà une alternative plus avantageuse. Cela s'applique de la même manière lorsque l'on recherche le minimum en échangeant le rôle d'alpha et bêta. (Voir exemple en Annexe 15)

Cette technique a eu un rôle significatif dans l'optimisation de notre algorithme, car elle permet d'éviter d'évaluer et de parcourir un grand nombre ce qui résulte en un gain de temps qui peut varier entre 20 et 30 secondes (avant l'utilisation des threads).

# Conclusion

Au cours de ce rapport, nous avons pu aborder les différents aspects de la création du jeu du Puissance 4 à travers la création d'une interface graphique. Nous avons ensuite abordé la création du Jeu du puissance 4 en mettant en place l'interactivité utilisateur machine. Nous avons également découvert la notion du machine learning avec l'entraînement beaucoup trop chronophage. Enfin, nous avons découvert la notion des arbres à travers l'implémentation de l'algorithme minimax qui permettaient la simulation des coups en avance.

Nous avons également souligné l'importance de la méthode d'évaluation pour déterminer le meilleur coup possible à jouer. Nous avons vu que cette méthode pouvait être complexe à mettre en place surtout pour être adaptative à la puissance.

Enfin, nous avons exploré deux techniques d'optimisation, l'utilisation de threads pour effectuer plusieurs tâches en parallèle et l'élagage alpha-bêta qui permet un parcours des arbres optimisé.

En somme, nous avons aussi vu pourquoi nous avons choisi l'algorithme minimax en dépend du machine learning.

Cependant, nous avons vu que le machine learning pouvait être utile et vraiment optimisé pour le début de partie contrairement au minimax qui est plus chronophage lors de l'exécution de programme en début de partie.

Ainsi, pour aller plus loin, il nous faudrait essayer de pouvoir combiner les deux algorithmes en utilisant l'entraînement du machine learning lorsque cela est possible et sinon on utilise le minimax dans le cas contraire ce qui permettrait aussi à notre algorithme de s'améliorer au fil des parties.

# Bibliographie

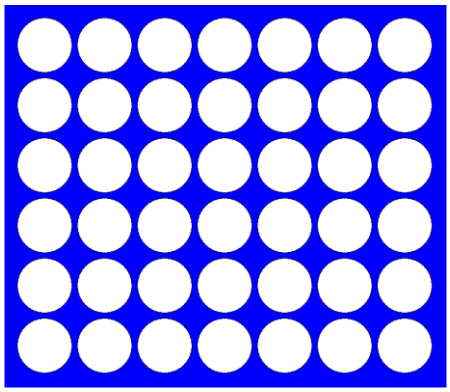
| A Propos | Lien | Type | Auteur |
|---|---|---|---|
| Python | 3.11.2 Documentation (python.org) | Documentation | |
| Tkinter | tkinter — Interface Python pour Tcl/Tk — Documentation Python 3.8.16 | Documentation | |
| Classe | 9. Classes — Documentation Python 3.11.2 | Documentation | |
| Exemple d'algorithme de puissance 4 « parfait » | Solveur de Puissance 4 (gamesolver.org) | Site web | Pascal Pons |
| Exemple d'algorithme puissance 4 mêlant machine-learning et algorithme minimax | Creating the PERFECT Connect 4 A.I. - YouTube | Vidéo YouTube | Code Bullet |
| Explication Minimax et élagage alpha-bêta | Algorithme Mini Max Elagage Alpha Beta - YouTube | Vidéo YouTube | Frederic Boissac |
| Récursivité | Démystification de la récursivité en Python (tutsplus.com) | Doc | Esther Vaati |
| Explication Machin learning | Le Machine Learning c'est quoi ? - YouTube | Vidéo YouTube | Defend Intelligence |
| Présentation technique Machine learning | Machine Learning : Définition, fonctionnement, utilisations (datascientest.com) | Site web | |

# Annexe

Annexe 1 :

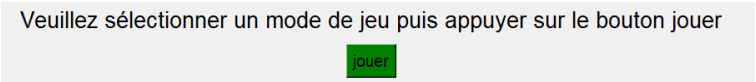
Puissance 4

Annexe 2 :

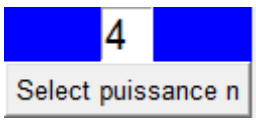


Annexe 3 :

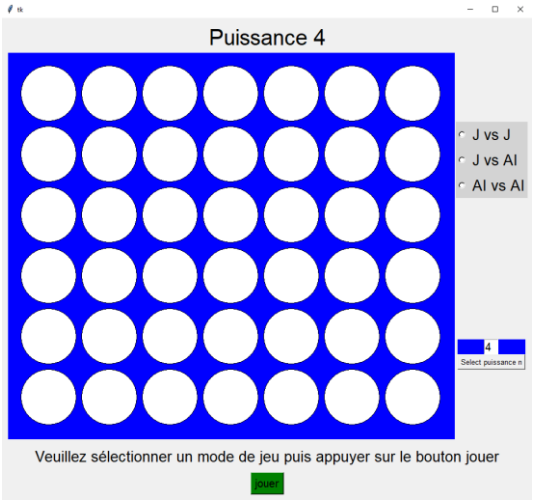

Annexe 4 :



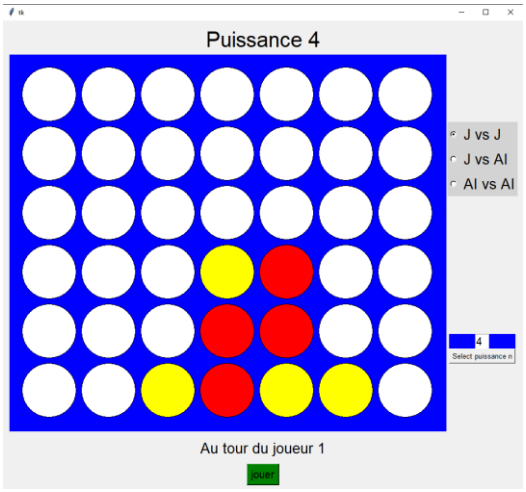

Annexe 5 :



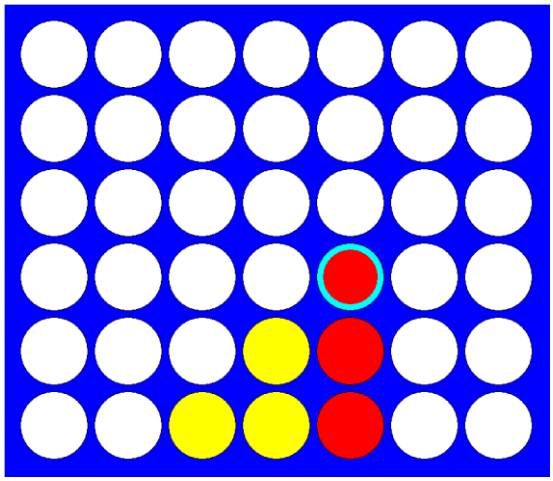
Annexe 6 :



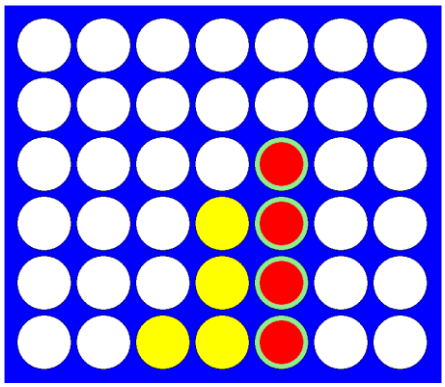

Annexe 7 :



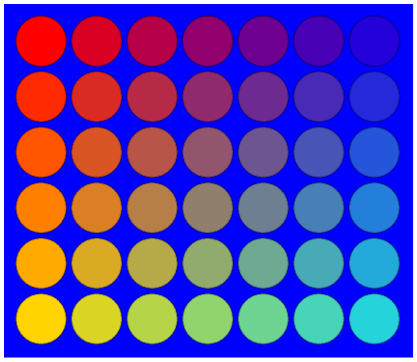

Annexe 8 :



Annexe 9 :



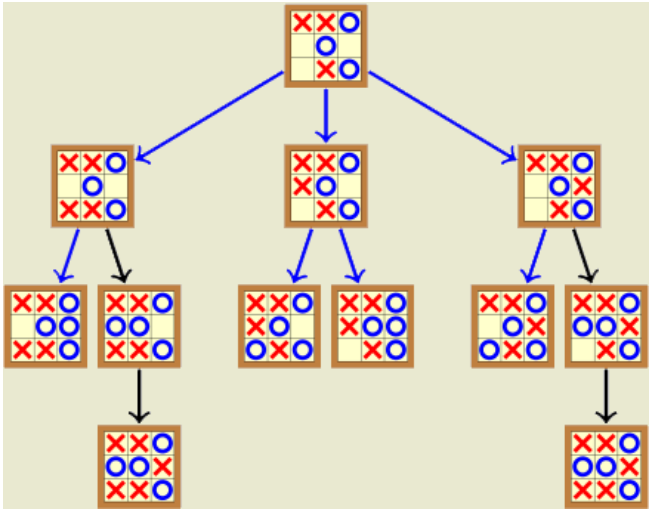
Annexe 10 :



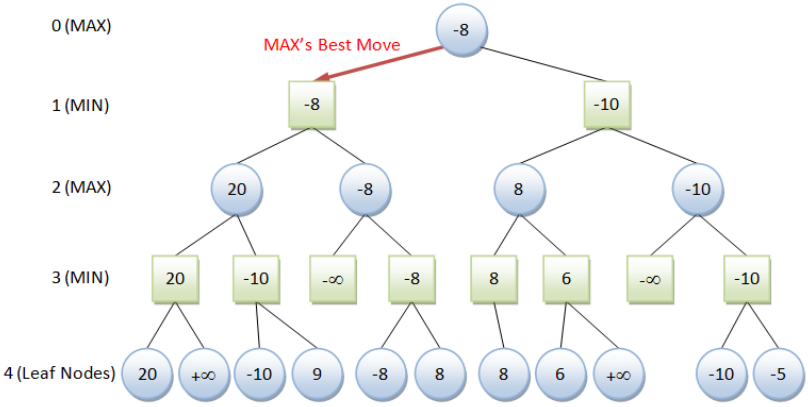
Annexe 11 :

| | board | action | eval | nb |
|---|---|---|---|---|
| 1 | | 0 | -0.0235294117647059 | 170 |
| 2 | | 1 | 0.140845070422535 | 142 |
| 3 | | 2 | 0.125628140703518 | 398 |
| 4 | | 3 | 0.67063020214031 | 3364 |
| 5 | | 4 | 0.167701863354037 | 644 |
| 6 | | 5 | 0.238993710691824 | 159 |
| 7 | | 6 | -0.406015037593985 | 133 |

Annexe 12 :



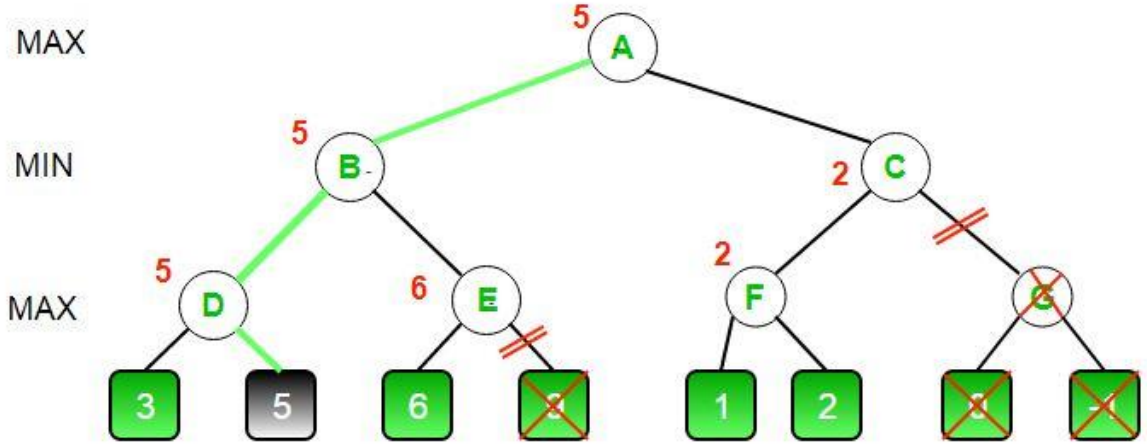
Annexe 13 :




Annexe 14 :

```
2 1.25999999900925426e-05
1 14.446405699999985
2 12.215661900000015
1 15.574115599999999
2 11.673852299999993
1 9.487224700000013
2 10.153933899999998
1 14.760155499999996
2 5.626575199999991
1 6.436002500000001
2 3.243169999999992
1 3.2608766000000173
2 2.5526787000000013
1 2.7828654999999856
2 4.903051099999999
1 1.3492516000000023
```

Annexe 15 :




## Compte rendu n°1 | 9/01/2023

Lors de la première séance, nous avons tout d'abord réfléchi à l'interface du jeu,

Nous avons pensé à utiliser un canvas sur lequel la partie du puissance 4 se déroulera.

Nous avons aussi imaginé une checkbox sur le côté qui permettra de sélectionner le mode de jeu. (joueur contre joueur ou contre l'IA et IA contre IA)

Ensuite, nous avons pensé à mettre un label en dessous de la grille pour indiquer le tour du joueur et pour finir un bouton "jouer" pour lancer le mode de jeu que l'on veut.

Nous avons imaginé avec le canvas que l'on pourra cliquer dessus pour jouer en détectant la colonne grâce à la coordonnée x du clique que l'on récupérera.

Nous avons donc commencé à construire le canvas en utilisant la POO en créant un objet jeu qui va gérer à la fois l'initialisation de l'interface graphique, mais aussi le déroulement du jeu.

Pour l'instant, nous avons mis en place un canvas juste bleu avec les checkbox sur le côté.

Nous avons en résolu aussi un premier problème avec les checkbox qui était déjà coché, le problème venait juste du fait que nous n'ayons pas assigné l'objet jeu lors du lancement ce qui provoquait un problème.

## Compte rendu n°2 | 16/01/2023

Comme vous avez pu le voir ce matin l'interface graphique de base est quasiment finie avec le jeu derrière,

Il nous reste juste deux problèmes à régler avec les cercles verts pour indiquer les pions gagnants, mais aussi le label pour indiquer le tour du joueur qui ne se met pas à jour,

Sinon la semaine prochaine je pense que nous pourrons finir cela et commencer à travailler sur l'IA qui se basera sur le principe du minimax comme j'ai pu vous l'expliquer ce matin

## Compte rendu n°3 | 23/01/2023

Comme vous avez pu le voir ce matin, l'interface graphique est quasiment terminée (sauf dans le cas où nous réussiront à implémenter un système afin de jouer aux différentes puissances 4,3,2, ...),

Nous avons ce matin aussi réglé les problèmes au niveau des labels, maintenant, ils se mettent bien à jour, avec un ajout de cercle bleu autour du dernier pion joué.

Il reste cependant un problème au niveau de la mise à jour du canvas lors du mode AI vs AI où la partie se déroule et nous donne le résultat à la fin, problème qu'il faudra régler.

De plus, ce matin, l'architecture de l'IA a été créée et est fonctionnelle, elle reste cependant largement améliorable et il faudra aussi que nous vous présentions comment celle-ci fonctionne.

## Compte rendu n°4 | 30/01/2023

Ce matin, nous avons mis en place le système des différentes puissances de n suite à votre demande de la semaine dernière,

Cependant, pour l'instant, les différentes puissances ne fonctionnent que pour le jeu de base, (on peut jouer dans tous les modes juste l'IA va tout le temps jouer comme si c'était un puissance 4)

Maintenant, il nous reste à adapter l'IA pour qu'elle puisse réfléchir suivant les différentes puissances, cela nous donnera aussi l'occasion d'améliorer notre méthode d'évaluation,
Pour vérifier si notre fonction d'évaluation est meilleure, nous pourrons notamment par la suite faire jouer les deux IA contre elle grâce aux modes IA vs IA prévu à cet effet.

De plus, nous vous avons aussi présenté la manière dont l'algorithme minimax fonctionner grâce à un schéma.

## Compte rendu n°5 | 6/02/2023

Ce matin, suite à la mise en place le système des différentes puissances de n, nous avons commencé à réadapter notre algo minimax pour qu'il puisse jouer suivant les différentes puissances,

Pour le moment, notre algorithme fonctionne pour toutes les puissances seulement il n'est vraiment performant que pour les puissances 2 et 3, mais continue de perdre contre notre précédent algorithme pour 4.

Ce matin, nous avons aussi corrigé plusieurs bugs comme la correction d'un bug lors de la sélection des puissances et maintenant lors du changement de la puissance le titre change aussi.

Nous avons aussi un bug qui faisait que nous pouvions lancer une partie sans avoir choisi de mode de jeu ce qui n'était pas normal.

## Compte rendu n°6 | 13/02/2023

Ce matin, nous vous avons présenté le fonctionnement de l'algorithme minimax et comment celui-ci nous permettait d'évaluer la meilleure colonne dans laquelle jouer.

Nous avons aussi amélioré notre algorithme puissance n pour jouer selon n'importe quelle puissance (2 - 6),

Il est maintenant efficace pour toutes les puissances, grâce à la correction d'un bug ce matin où nous nous sommes rendu compte que nous évaluons toujours la grilla initiale et non les simulations.

Par la suite, nous avons donc testé cet algorithme contre notre ancien, les victoires sont souvent 50/50, cela dépend aussi de celui qui commence.

Enfin, nous avons fini par tester si notre IA était vraiment plus performante lorsque nous augmentions la profondeur ce qui était le cas en la testant contre elle-même avec une profondeur inférieure.

Nous avons aussi corrigé un bug qui faisait que nous pouvions cliquer dans une 8e colonne en cliquant tout à droite ce qui est maintenant impossible.

## Compte rendu n°7 | 27/02/2023

Ce matin, nous avons rajouté une animation lors d'une égalité et corrigé un bug du label lorsqu'une égalité se produisait.
Puis dans l'optique d'optimiser le temps d'exécution de notre algorithme nous avons commencé à essayer de réaliser des threads en python afin de lancer les 7 première recherches en même temps de notre algo minimax.

## Compte rendu n°8 | 6/03/2023

Ce matin, nous avons mis en place les threads dans notre algorithme, nous avons pu remarquer une amélioration du temps d'exécution de plusieurs secondes notamment pour les simulations les plus longues du début, un gain de 5-10 secondes.

Nous avons aussi découvert une nouvelle piste pour gagner du temps avec l'élagage alpha-bêta qui consiste à supprimer les branches de l'arbre inutile.

Nous vous avons aussi présenté une autre manière de penser un algorithme pour le puissance 4 avec du machine learning, mais celui-ci était vraiment long pour qu'il devienne exploitable, c'est pourquoi nous avions choisi d'utiliser l'algo minimax.

## Compte rendu n°9 | 13/03/2023

Finalisation du projet avec ajustement des conventions et mise en place de l'optimisation alpha-bêta et commencement de l'écriture du rapport.


## Résumé

Notre projet avait pour objectif de concevoir une version informatique du Puissance 4 à l'aide de Python et de la bibliothèque tkinter pour l'interface graphique.

Nous avons ensuite mis en place un algorithme minimax pour créer une intelligence artificielle capable de jouer contre un joueur humain.

Cet algorithme repose sur le principe des arbres avec la récursivité et simule un certain nombre de coups à l'avance pour déterminer le meilleur coup possible.

Nous avons optimisé l'exécution de cet algorithme en utilisant le multithreading pour lancer plusieurs tâches en parallèle, ainsi que l'élagage alpha-beta pour supprimer les branches inutiles dans l'arbre.

Nous avons également exploré la possibilité d'utiliser un algorithme de machine learning pour entraîner notre IA à jouer au Puissance 4.

Cependant, cette approche s'est avérée trop complexe et chronophage pour notre projet, donc nous avons choisi de ne pas retenir cette technique.

## Mots-clés

Puissance 4, intelligence artificiel, algo minimax, threads, élagage alpha beta, machine learning, interface graphique, tkinter, python, récursivité

## Abstract

The aim of our project was to design a computer version of connect 4 using Python and the tkinter library for the graphical interface.

Then we implemented a minimax algorithm to create an artificial intelligence wich is able to play against a human player.

This algorithm is based on the tree principle with recursion with simulating a number of moves in advance to determine the best possible move.

We optimised the execution of this algorithm by using mutlithreading to run several tasks in parallel, as well as alpha-beta pruning to remove unnecessary branches in the tree.

We also explored the possibility of using a machine learning algorithm to train our AI to play connect 4.

However, this approach proved to be too complex and time consuming for our project, so we choose to not use this technique.

## Keywords

Connect 4, artificial intelligence, minimax algorithm, alpha beta pruning, machine learning, GUI, tkinter, python, recursion


| Encadrant académique | Étudiants |
| --- | --- |
| *Jean-Charles BILLAUT* | *Noah LOUET*<br>*Mattéo FOUGERAY* |