**Enzo DE SOUSA**

**Samy PELISSON**

**ESILV**

**Rapport du projet Datascience & IA :**

**Morpion & Puissance 4**

Sommaire

[Introduction 2](#_Toc103023072)

[I. Choix 3](#_Toc103023073)

[1. Morpion 3](#_Toc103023074)

[2. Puissance 4 4](#_Toc103023075)

[II. Implémentation et code 7](#_Toc103023076)

[1. Morpion 7](#_Toc103023077)

[2. Puissance 4 12](#_Toc103023078)

[Conclusion 18](#_Toc103023079)

[Références 19](#_Toc103023080)

Introduction

Il nous a été demandé dans le cadre de notre première année du cycle ingénieur à l’ESILV de créer un Morpion et un Puissance 4, tous deux fonctionnant en ligne de commande et permettant de combattre une IA. Ce projet devait donc nous permettre d’utiliser diverses compétences que l’on a pu acquérir au cours du 1er et 2ème semestre. Il devait être effectué par groupe de deux ou trois. Le nôtre est alors composé de Samy PELISSON et d’Enzo DE SOUSA. Le langage imposé était le Python et notre code devait être capable de tourner sur Google Colab et allait être utilisé afin d’affronter les IA des autre groupes de TD.

Certaines règles devaient être respectées, nous devions d’abord commencer par créer un Morpion et appliquer l’algorithme du min-max. Pour le Puissance 4, il fallait appliquer un élagage en plus, il s’agit ici d’alpha-beta.

Concernant le Morpion, aucune restriction ou changement dans les règles nous ont été signalé, il s’agit donc d’un classique 3x3 pouvant se jouer avec un joueur X et un joueur O.

Pour le Puissance 4, il fallait pouvoir jouer sur 12 colonnes et 6 lignes car sur la version classique il était trop facile de pouvoir gagner en jouant au milieu directement. A savoir que l’entrée par l’utilisateur devait se faire en entrant un chiffre de 1 à 12, sélectionnant la colonne dans laquelle l’utilisateur souhaite placer son pion.

Nous allons voir dans un premier temps les différentes problématiques auxquelles nous avons dû faire face ainsi que les choix prit en conséquence par rapport à celles-ci. Nous allons par la suite nous pencher sur le code, détaillant son fonctionnement ainsi que certaines parties plus complexes à comprendre.

1. Choix

Tout d’abord concernant les deux projets, nous avons opté pour une programmation en module. Nous aurions pu nous contenter de faire tous le projet dans un même fichier Python, mais il nous semblait préférable de garder un minimum de lisibilité et de ségréguer un maximum les tâches.

Dans les deux projets, nous avons opté pour des tableaux (array) avec le module numpy. Puisque les éléments sont stockés à la suite dans la mémoire, il sera plus rapide de lire ces éléments. En contrepartie, il sera plus long de créer notre tableau.

1. Morpion

Il faut savoir dans un premier temps que dans le morpion et en utilisant l’algorithme du min-max (avec présence de l’élagage alpha-beta ou non), il n’est pas possible de gagner. Les seules issues possibles sont pour le joueur la défaite ou l’égalité. A partir de ce constat, il est très simple de tester le fonctionnement de notre algorithme.

Voici les différents module au sein du projet Morpion :

Une image contenant texte

Description générée automatiquement

Explication des modules :

* Action : un module permettant de simplement représenter un objet ayant des coordonnées en 2D x et y ;
* Board : il s’agit du module représentant notre tableau de jeu 3x3, permettant d’y placer des pions et obtenir le gagnant ;
* Main : fichier contenant le programme principal de notre jeu, il met en lien les différents modules et permet à l’utilisateur d’entrer ses choix ;
* Min\_max\_algorithm : module contenant notre algorithme du min-max ;
* Utilities : fichier contenant des fonctions utiles.

Le jeu se joue donc en ligne de commande et se présente de la façon suivante :

Une image contenant texte

Description générée automatiquement

Nous avons opté pour une entrée de la part de l’utilisateur en coordonnées commençant de 0 à 2, séparés par une virgule. Il s’agit ici d’un choix propre à nous puisque dans le monde de l’informatique il est très courant de commencer à compter à partir de 0.

Étant donné que par la suite nous avons découvert un élément permettant d’accélérer certaines de nos fonctions (expliqué dans la partie suivante sur le puissance 4 et les matrices de convolution), nous aurions pu (dans l’idéal) mettre en place cette solution pour le Morpion. Encore une fois, nous avons fait le choix de laisser le projet comme ceci, d’une part parce que le Morpion n’avait pas pour vocation d’affronter les autres IAs codées par les personnes du groupe de TD mais aussi parce que nous n’avions pas le temps.

1. Puissance 4

Le Puissance 4 va changer un petit peu dans son implémentation mais il va globalement reprendre la même structure que pour le Morpion :

Une image contenant texte

Description générée automatiquement

Étant donné que la plupart des modules ont la même utilité que dans le puissance 4 et que seulement leur implémentation change, je vais me contenter d’expliquer ce qu’est le module Bitboard.

Ce module a en fait la même utilité que le module Board, c’est-à-dire qu’il contient notre représentation du jeu puissance 4 mais sous une forme bien particulière : il utilise des bits au lieu d’utiliser des tableaux en deux dimensions.

Sans trop rentrer dans les détails, il est en effet possible de représenter le jeu Puissance 4 (et d’ailleurs le Morpion aussi) sous forme binaire. Cette implémentation, un peu plus difficile à mettre en œuvre a l’avantage d’être bien plus rapide qu’une implémentation en deux dimensions. Ceci vient tout simplement du fait que nous utilisons une chaine de caractère (qui en mémoire est un tableau de caractère) mais aussi parce qu’elle permet d’effectuer des vérifications (qui est le gagnant, quelle sont les actions possibles) bien plus rapidement.

Malheureusement et bien que très proche de réussir cette implémentation, nous n’avons pas eu le temps nécessaire pour la mettre en œuvre. Nous avons décidé de laisser ce module dans le code, mais de partir sur une implémentation basique en deux dimensions, bien que plus lente.

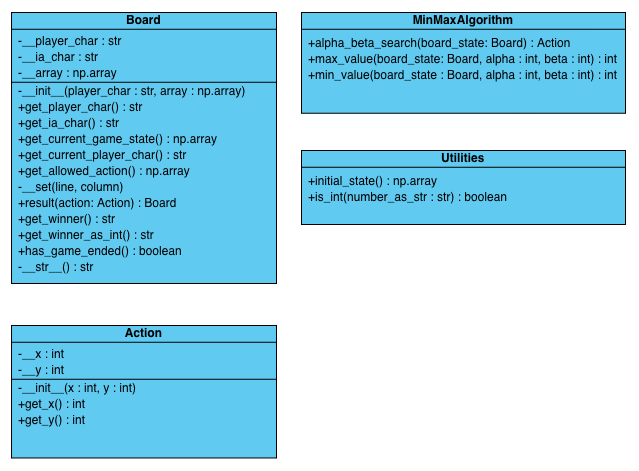
Un autre choix que nous avons dû faire ici est concernant la façon de vérifier le vainqueur dans le Puissance 4. Comme expliqué plus haut, il est beaucoup plus lent d’utiliser des tableaux en deux dimensions et nous nous sommes heurtés à un mur concernant le temps d’exécution. Nous avons donc trouvé un moyen plus élégant, prenant moins de place à implémenter et utilisant des propriétés mathématiques : les matrices de convolution. Nous détaillerons le fonctionnement dans notre prochaine partie, le code et son implémentation.

Étant donné que les deux projets se ressemblent énormément, nous aurions pu interfacer le projet de sorte à avoir une classe mère reprenant les fonctionnement communs aux deux jeux, et en redéfinissant les mécanismes nécessaires dans des classes filles. Cependant, c’est une finition que nous n’avons pas eu le temps d’effectuer.Implémentation et code

Nous allons dans cette partie étudier les implémentations des principales fonctions de notre code ainsi que du fonctionnement global du programme.

1. Morpion

Voyons dans un premier temps un court diagramme de classe permettant de prendre conscience des différentes fonctions au sein de notre jeu.

  
Les fonctions les plus utilisées seront celles appelées récursivement au sein de la classe MinMaxAlgorithm (max\_value et min\_value), qui, elles-mêmes en appellent d’autres. Regardons l’implémentation de cette classe et les fonctions appelées couramment dans la classe Board.

Une image contenant texte

Description générée automatiquement  
alpha\_beta\_search est la fonction qui sera appelée dans la classe main afin d’obtenir les coordonnées dans lesquels l’IA devrait idéalement placer son pion. Cette fonction est basée sur le pseudo-code qui nous a été fournit afin de commencer le projet du Morpion, elle retourne simplement la meilleure action à effectuer pour l’IA en appliquant l’algorithme du min-max.

L’algorithme cherche à maximiser les gains pour un des deux joueurs et à le minimiser pour l’autre. Ainsi, en considérant que la racine est un nœud « maximum », alors les prochains nœuds fils devront être des nœuds « minimum », et ainsi de suite. Dans notre cas, l’IA cherche à maximiser ses gains et cherche à minimiser les gains du joueur.

Une image contenant texte

Description générée automatiquement  
Comme nous avons pu le voir dans le code, la fonction alpha\_beta\_search va commencer par appeler min\_value sur toutes les actions possibles du joueur (puisque l’IA cherche à minimiser les gains du joueur).

Ainsi, voici les remarques que nous pouvons en tirer :

* La fonction has\_game\_ended() dans le module Board est exécutée de multiple fois afin de vérifier si il y a un vainqueur, elle a donc tout intérêt à être rapide (cf. remarques énoncés dans nos choix concernant le Puissance 4 et le Bitboard qui aurait pu permettre, appliqué au Morpion, d’accélérer nettement l’exécution de celui-ci).
* Les prochains nœuds seront donc des nœuds que l’IA cherchera à maximiser, c’est donc pour cela que l’on appelle max\_value() sur les actions possibles.

A titre indicatif, voici maintenant l’implémentation de max\_value(), qui fonctionne à l’opposé de min\_value() :

Une image contenant texte

Description générée automatiquement  
Voyons maintenant les fonctions exécutées le plus fréquemment dans notre classe Board :

Une image contenant texte

Description générée automatiquement

Cette fonction vérifie simplement si toutes les cases dans lesquels il est possible de faire une action ont été remplies ou bien s’il y a un gagnant (get\_winner()).

Ainsi, la fonction get\_winner() sera elle aussi exécutée de nombreuse fois (et doit être rapide). Étant donné que nous avons découvert les matrices de convolution après ce projet du Morpion, nous avons décidé d’opter pour les propriétés des sets en Python afin de vérifier la présence d’un gagnant :

Une image contenant texte

Description générée automatiquement

En effet, pour une ligne, une colonne (qui est simplement la transposée d’un vecteur ligne) ou bien les diagonales, s’il y’a présence de plus d’un élément dans le set alors le vecteur en question ne comporte pas de gagnant (à condition que cet élément ne soit pas l’élément vide).

Rappel : En Python, un set est un ensemble de données non ordonnées et non redondantes.

Voyons maintenant, la dernière fonction : get\_allowed\_actions().

Une image contenant texte

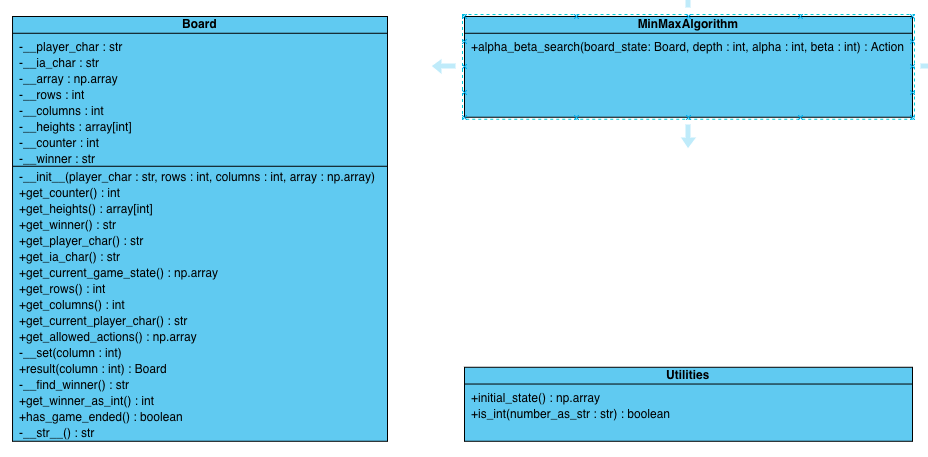
Description générée automatiquement

Cette fonction permet de retourner sous forme de tableau numpy toutes les cases qui ne sont pas vides.

Comme expliqué dans la première partie, nous avons ici fait le choix d’utiliser des tableaux numpy puisqu’ils sont plus rapide lors du parcours des actions dans les fonctions min-max.

1. Puissance 4

Voici le diagramme de classe concernant le projet du Puissance 4.



Sans trop m’attarder sur les éléments qui restent les mêmes globalement (essentiellement l’algorithme du min-max que nous avons réduit à une seule et unique fonction mais que nous allons quand même montrer ainsi que de la classe Board et son exécution dans le main), voici ce qui a changé :

Une image contenant texte

Description générée automatiquement

Notre algorithme alpha\_beta\_search() retourne toujours l’action la plus idéale a effectuer pour un jouer donné, elle se fait maintenant en une seule fonction, elle aurait pu se décomposer en plusieurs fonctions mais le fonctionnement est le même et le passage de min a max se fait grasse a un bouléen (should\_maximize).

La grande différence avec le min\_max du morpion est l’apparition de la profondeur (depth) qui va nous permettre de limiter la profondeur de la recherche récursive, ce n’est pas un choix mais une contrainte étant donné le nombre de possiblitées extrêmement élevé d’un jeu de Puissance 4.

La méthode has\_game\_ended() reste la même que dans le Morpion, cependant, nous avons rajouté dessus un cache sur le gagnant afin de ne pas réeffectuer les opérations une nouvelle fois et nous avons aussi ajouté le nombre maximum de pions disponibles (42).

Une image contenant texte

Description générée automatiquement

**Une image contenant texte

Description générée automatiquement**De l’autre côté, la méthode get\_winner() (qui est ici renommée en \_\_find\_winner()) voit aussi son implémentation changer :

Comme nous avons pu l’énoncer plus haut avec les matrices de convolution, il s’agit d’une solution trouvée en ligne que nous avons décidé d’intégrer à notre projet.

Dans notre cas, nous définissons ce qu’on va appeler les noyaux. Il s’agira pour nous de nos conditions de victoire :

Une image contenant texte

Description générée automatiquement

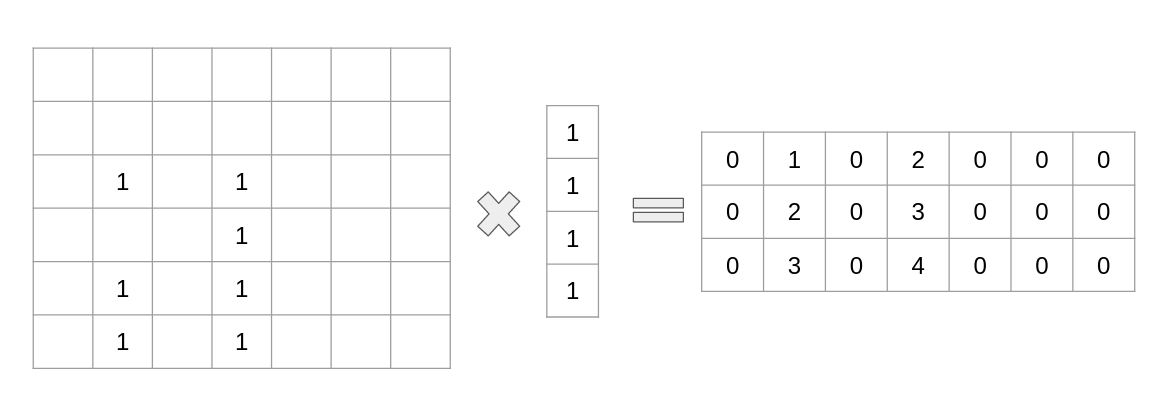
Ainsi, nos conditions de victoires sont les suivantes :

* Un vecteur ligne de 4 éléments
* Un vecteur colonne de 4 éléments
* Une matrice ayant pour diagonale que des 1 en partant du haut à gauche
* Une matrice ayant pour diagonale que des 1 en partant d’en haut à droite

Nous stockerons ces conditions dans une liste (ici « DETECTION\_KERNELS ») afin d’itérer dessus dans \_\_find\_winner().

Finalement, la matrice de convolution (ici la fonction convolve2d) va prendre notre matrice de jeu (le Board, qui, je le rappelle, est une liste de tableau en 2D) et pour chaque élément de celui-ci, va faire la somme de ces éléments en reprenant la forme du noyau (matrice, vecteur colonne, vecteur ligne) multiplié par l’élément à la même position dans le noyau.

Pour illustrer au mieux ce qui se passe, voici un exemple :



Désormais, il ne nous reste plus qu’à regarder si pour chacun des joueurs avec son propre tableau de jeu isolé (sans les pions de l’adversaire) nous trouvons un 4 (notre condition de victoire) dans la matrice que nous retourne convolve2d. Si c’est le cas, alors le joueur isolé en question a aligné 4 pions et la victoire lui revient donc.

Concernant get\_allowed\_actions(), le principe reste le même : on doit pouvoir avoir les colonnes dans lesquels un joueur est capable de jouer.

Une image contenant texte

Description générée automatiquement

Pour se faire, rien de plus simple, nous vérifions si la colonne n’est pas remplie et si c’est le cas nous l’ajoutons à la liste que nous allons retourner.

Conclusion

Ce projet aura été l’occasion pour nous de mettre en œuvre beaucoup de compétences que nous avons pu acquérir au cours des différents modules : principalement Datascience & IA et Python.

Malgré les problèmes rencontrés (manque de temps, bugs…) ce projet fut un réel plaisir à faire, il nous a permis de découvrir certain.e.s outils ou structures dont nous n’avions pas connaissance (matrice de convolution, bitboard).

Nous aurions aimé avoir la possibilité d’intégrer du polymorphisme ainsi qu’un Bitboard, nous sommes certains que rendre le projet homogène l’aurait rendu plus propre. Une structure de donnée adaptée aurait permis d’augmenter grandement les performances de notre algorithme. Idéalement pour les performances, il aurait fallu faire cet algorithme en C/C++ mais le sujet nous imposait Python.

Références

<https://stackoverflow.com/questions/61072185/minimax-algorithm-for-tictactoe-in-python>

<https://stackoverflow.com/questions/29949169/python-connect-4-check-win-function>

<https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f>

<https://github.com/shigatena666/Projet> (il s’agit de notre dépôt sur GitHub)