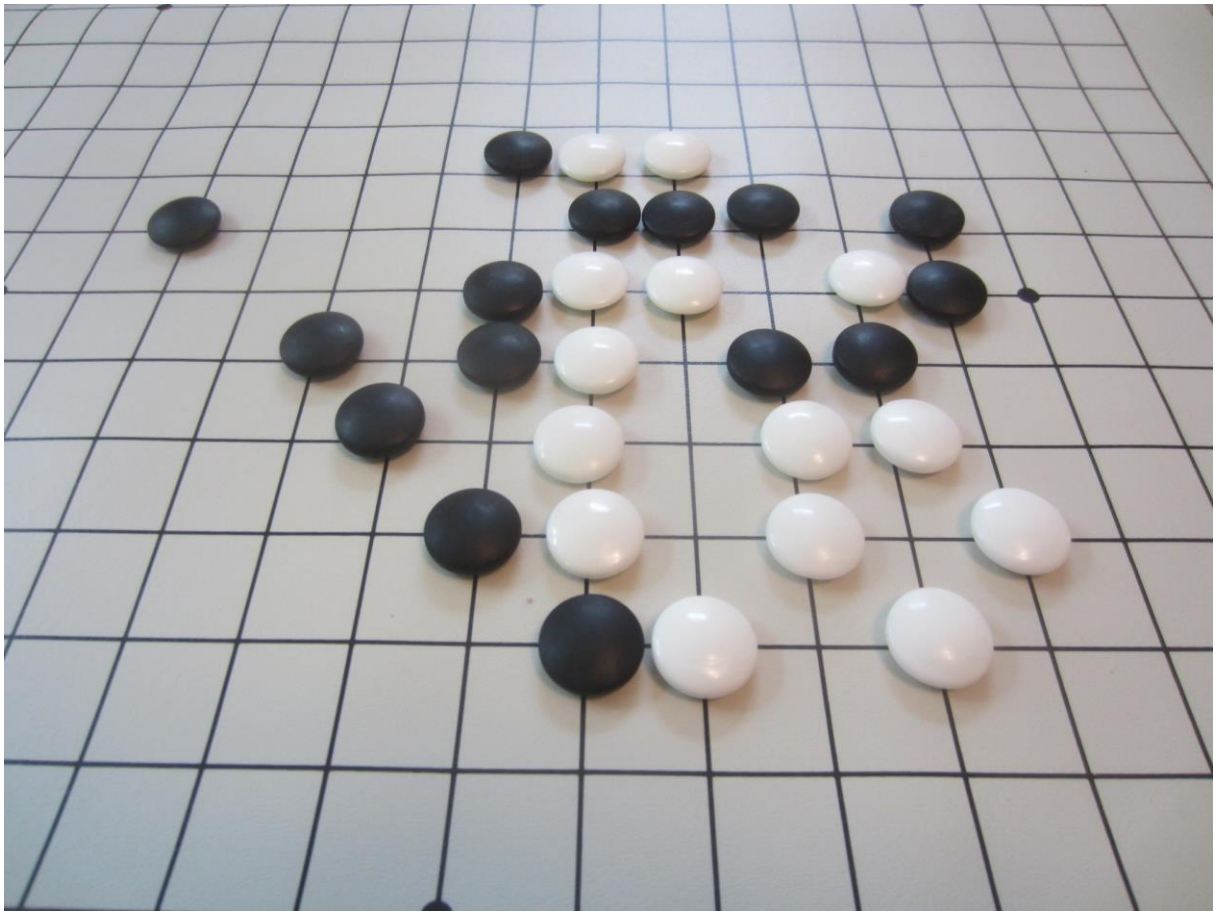
PROJET GOMOKU



SOMMAIRE

PRÉSENTATION DU PROJET
EXPLICATION DU « MINMAX »
EXPLICATION DU CALCUL DE POIDS
DIFFÉRENTES VERSIONS DU PROJET
PROBLÈMES RENCONTRÉS ET SOLUTIONS

PRÉSENTATION DU GOMOKU

Le Gomoku est le nom japonais du jeu de plateau chinois Wǔzi qí. Gomoku signifie littéralement « alignement des cinq pions ». Le but du jeu consiste à aligner 5 pions sur les intersections d'un plateau de jeu de go. Découvert au siècle dernier par les anglo-saxons, le gomoku a des millions d'adeptes en Extrême-Orient (Chine, Corée, Japon).

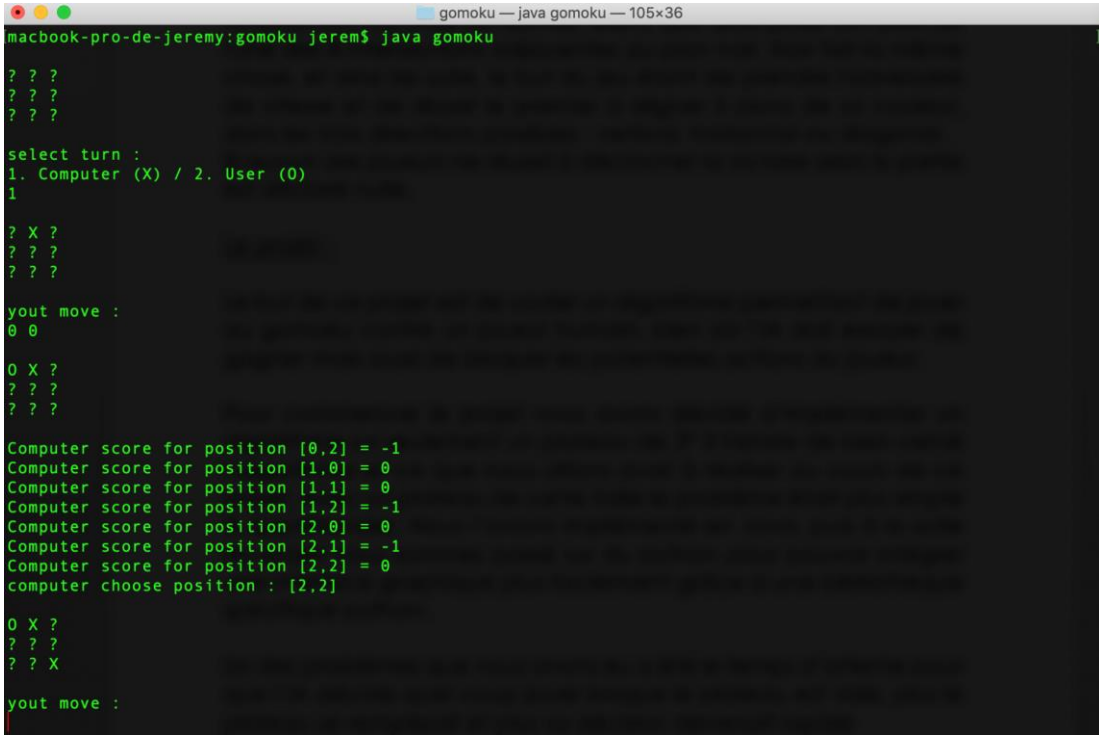
Les règles :

Le gomoku se joue sur un plateau quadrillé de 19 lignes horizontales et 19 lignes verticales formant, comme un jeu de go, 361 intersections. Le nombre de pions est toutefois nettement inférieur, puisque les joueurs n'en reçoivent que 60, qu'ils posent un à un et à tour de rôle sur les intersections.
Le joueur qui a choisi ou obtenu par tirage au sort les pions noirs jouent toujours le premier en plaçant son premier pion sur l'intersection centrale du damier. Le joueur adverse (Blanc) doit alors poser son pion sur l'une des 8 intersections adjacentes au pion noir. Noir fait la même chose, et ainsi de suite, le but du jeu étant de prendre l'adversaire de vitesse et de réussir le premier à aligner 5 pions de sa couleur, dans les trois directions possibles : verticale, horizontale ou diagonale. Si aucun des joueurs ne réussit à décrocher la victoire, la partie est alors déclarée nulle.

Le projet :

Le but de ce projet est de créer un algorithme capable de jouer au gomoku contre un joueur humain. Pour cela, l'IA doit essayer de gagner en alignant ses pions mais doit aussi bloquer les potentielles actions du joueur adverse.
Pour commencer le projet nous avons décidé d'implémenter un algorithme d'abord sur un plateau de dimension 3*3 afin de bien comprendre le principe de l'algorithme et ce que nous allions avoir à réaliser au cours de ce projet. Avec un plateau de cette taille, le problème était plus simple à appréhender. Nous avons d'abord implémenté l'algorithme en Java, puis nous sommes ensuite passés au langage Python pour pouvoir intégrer une interface graphique plus facilement grâce à la bibliothèque spécifique Python Tkinter.

Nous avons créé un algorithme de type MinMax qui utilise une fonction d'évaluation pour donner un certain poids à chaque coup possible et qui définit un arbre de choix pour décider quel coup jouer.



*Plateau de 3*3 avec indication des choix de l'ordinateur*

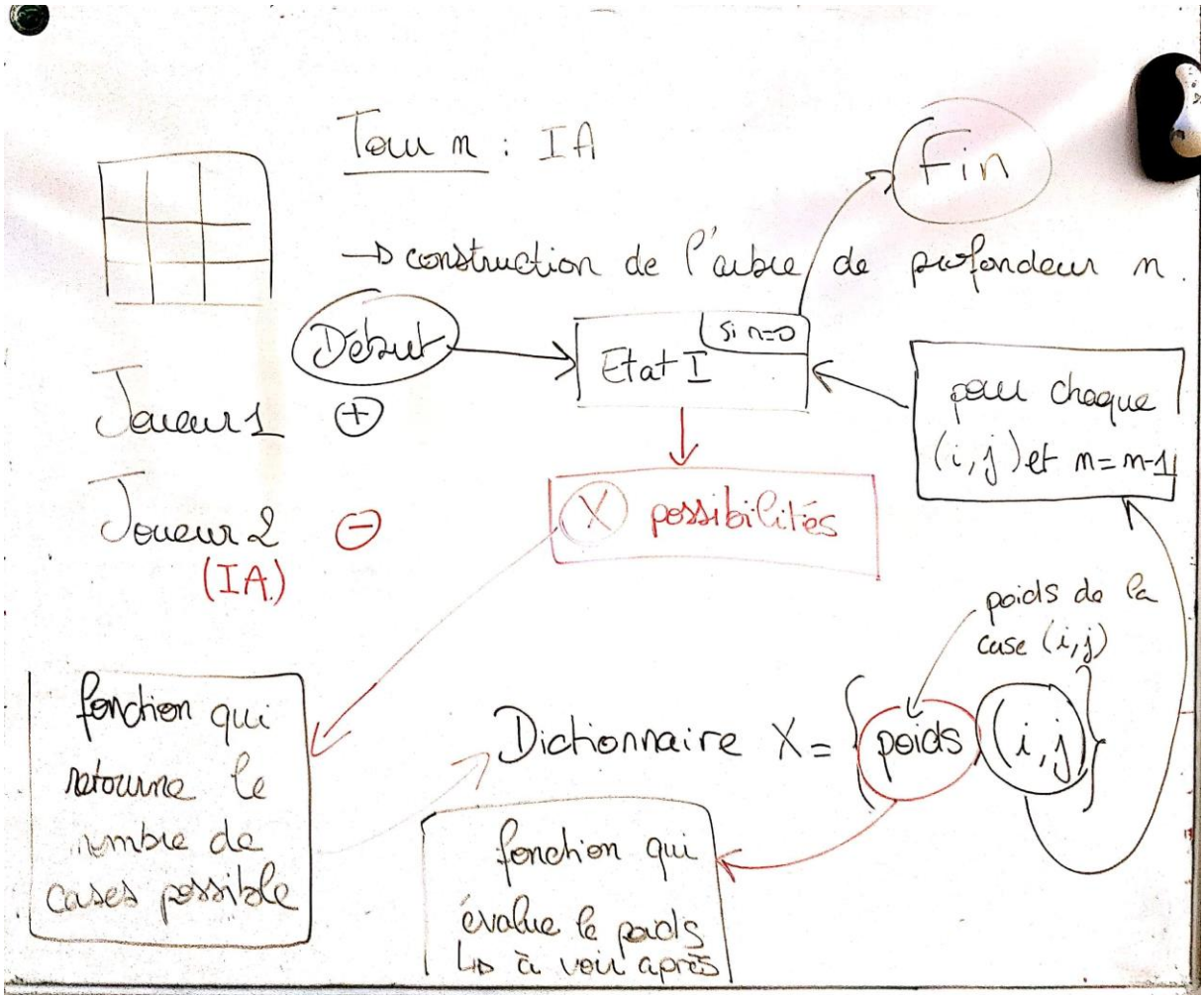
Un des problèmes que nous avons eu a été le temps d'attente pour que l'IA décide quel coup jouer lorsque le plateau est vide. Étant donné les nombreuses possibilités de coups, le temps de décision était long, mais plus le plateau se remplissait, et plus sa décision devenait rapide.

```
JOUEUR 2

Temps de rÈflexion : 36 secondes
 ____________________________________________
|-10 |-10 |-10 |-10 |-10 |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 |-10 |-10 |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 |-10 |-10 |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 | -1 |-10 |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 |-10 | 1  |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 |-10 |-10 |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 |-10 |-10 |-10 |-10 |-10 |
 ____________________________________________
|-10 |-10 |-10 |-10 |-10 |-10 |-10 |-10 |
 ____________________________________________
```

EXPLICATION DU « MINMAX »

Pour essayer de tirer un maximum de profit de cette unité nous nous sommes lancés quasiment sans base de code décrivant le « minmax ». Nous avons seulement regardé le modèle théorique de cette méthode. Modèle que nous avons schématisé comme le montre les photos ci-dessous

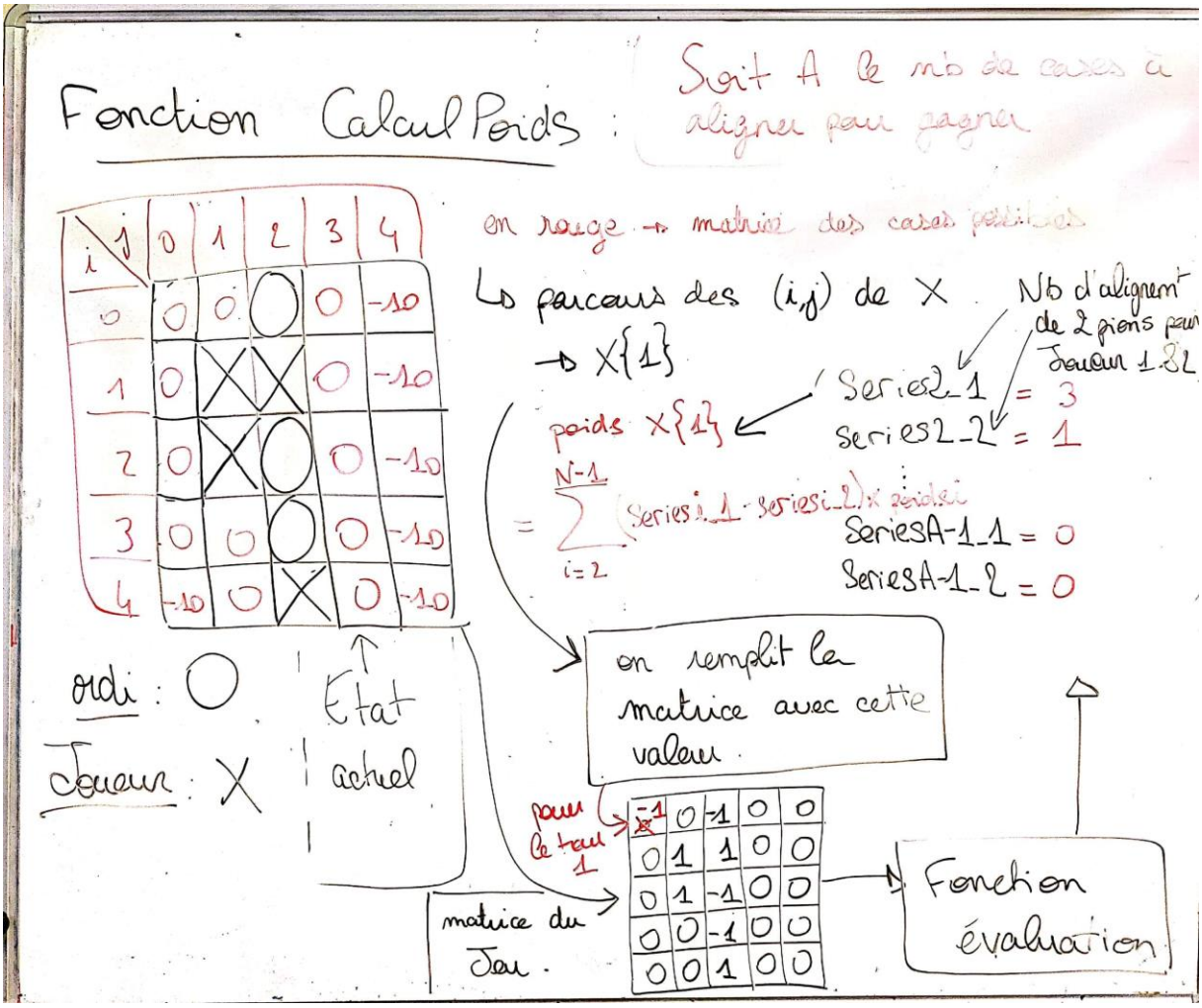


Dans un premier nous avons imaginé comment construire l'arbre en partant d'un état du jeu (correspondant à un certain avancement), il fallait bien évidemment construire une fonction permettant d'évaluer toutes les possibilités pour l'ordinateur à partir de cet état. Ensuite pour chaque possibilité trouvée, lui associer un poids, c'est-à-dire une valeur numérique qui en fonction de sa valeur permettra d'émettre des hypothèses dans les choix (voir la partir sur le calcul du poids).
Ensuite pour chaque possibilité, si la profondeur du calcul de l'arbre était supérieure à 1 alors il fallait simuler un coup supplémentaire, autrement dit pour chaque possibilité précédemment trouvée, exécuter les mêmes opérations que pour l'état initial ; et ainsi de suite en fonction de la valeur de la profondeur choisie. Nous nous sommes tout de suite rendu compte que c'est ce paramètre qui allait poser un problème dans l'équilibre d'un bon algorithme.

Effectivement la simulation d'un tour reste tout à fait raisonnable même pour un grand damier, cependant dès que l'on augmente d'un tour, le nombre de calcul est multiplié par lui-même (au minimum sauf cas extrêmes).
C'est donc pour cela que nous avons choisi une profondeur de 3 au début.

EXPLICATION DU CALCUL DE POIDS

L'évaluation du poids d'une possibilité représente une partie importante selon nous dans le bon fonctionnement de l'algorithme, puisque si le poids entre deux choix diamétralement opposés en termes de leadership de la partie se retrouve avec un poids similaire un gros problème est soulevé.
Pour notre cas l'évaluation du poids d'une possibilité se fait en deux temps dans un premier temps on récupère pour chacun des joueurs le nombre d'alignements de 2 jusqu'à N-1 (N représentant le nombre de pion à aligner pour gagner) en prenant soin de vérifier s'il y a blocage de ces alignements pour éviter les poids inutiles.
Une fois ces données récupérées nous avons appliqué une somme sur tous les



alignements pour chacun des joueurs avec pour chacun des alignements un « sous-poids » différents en fonction du nombre de pions alignés. Effectivement plus on se rapproche du nombre de pions à aligner pour gagner plus le poids doit être important pour pouvoir faire comprendre à l'algorithme qu'il doit vers ce chemin. Ensuite en fonction du joueur cette somme est multipliée par 1 pour le joueur 1 et par -1 pour le joueur 2 (l'ordinateur). Ainsi quand le poids global d'une possibilité est négatif cela signifie qu'à cet état du jeu le joueur 2 est dans une meilleure position que le

joueur 1. Bien entendu après réflexion on peut se dire que ce n'est pas une manière infaillible car il peut exister des techniques impliquant très peu d'alignements mais menant à une victoire certaine, cela fait partie des améliorations que nous pourrions apporter au jeu.
Par la suite dans l'une des versions nous avons introduit un système de blocage intuitif se basant sur ce système en effet un poids d'une valeur fonction du nombre de pions alignés par l'adversaire était sommé au poids global de ladite possibilité. En effet plus l'adversaire a de pions d'aligné plus l'ordinateur doit réagir pour le bloquer.

DIFFÉRENTES VERSIONS DU PROJET

La première version de l'algorithme avait plusieurs problèmes, comme le fait que la construction de l'arbre des possibilités ne s'arrêtait pas malgré que la victoire ait été détectée. Cela posait plusieurs problèmes puisque par la suite quand il fallait évaluer le meilleur coup possible en remontant l'arbre des possibilités le fait qu'un chemin de profondeur non nulle (donc qui ne correspond pas aux feuilles) contienne une victoire, toutes les possibilités suivantes reprenaient cette victoire ce qui créait donc un paradoxe dans certains choix.

Dans une deuxième version nous avons essayé d'améliorer ce problème en bloquant la construction de l'arbre à une certaine profondeur qui était définie quand une victoire était détectée dans un des chemins choisis.

Dans une troisième version nous avons ajouté en plus à l'IA un système de blocage, très simple basé sur le même système que le « MinMax » utilisé pour l'évaluation de la position. Pour ce faire nous avons attribué aux cases situées avant et après les alignements détectés ; si celles-ci étaient libres, un poids d'une valeur opposée à celle des pions alignés. Cela permet ainsi de créer une illusion dans l'algorithme, qui pense que cette case est plus importante.

Enfin dans une dernière version nous avons recréé une fonction permettant l'évaluation des diagonales dans le damier. En effet, celle qui était utilisé jusqu'à précédent était très efficace pour la détection de victoire et des alignements diagonaux. Néanmoins quand nous avons implanté un système de blocage la récupération des coordonnées où l'IA devait bloquer le joueur n'était pas possible.

PROBLÈMES RENCONTRÉS ET SOLUTIONS

Certains problèmes ont déjà été expliqués dans les parties précédentes, mais il reste quelques pistes d'améliorations. Une des premières qui n'est pas des moindres car elle permettrait la construction d'un arbre beaucoup moins coûteux. Elle peut également s'appeler l'élagage puisqu'elle consiste en la suppression ou le non calcul de certaines parties de l'arbre en fonction de conditions prédéfinies.
Dans un deuxième temps il faudrait trouver un équilibre entre la profondeur de calcul de l'arbre et les conditions de jeu, c'est-à-dire le nombre de pions à aligner pour gagner mais aussi la taille du damier.
Enfin lors de nos recherches nous avons découvert des algorithmes dits de « DeepLearning » et nous pensons que ce sont les meilleurs, mais nous n'avions pas le temps et les compétences pour pouvoir en développer un dans le temps imparti. Ces algorithmes consistent en une analyse du jeu de l'adversaire pour pouvoir jouer en conséquence et donc être plus performant. Cela permet également de se libérer de l'hypothèse que nous faisions en utilisant l'algorithme « MinMax » qui suppose qu'à chaque tour le meilleur coup est joué par chacun des joueurs.

INSTRUCTIONS

Par manque de temps nous n'avons pas pu implémenter la partie non graphique dans le programme utilisant la librairie « Tkinter », cependant nous vous fournissons les derniers fichiers dans lesquels nous avons tenté d'implémenter l'IA sans grand succès, juste pour que vous puissiez constater des programmes.
Dans un autre dossier il y aura le programme sans affichage graphique (seulement dans le terminal) qui a rencontré certains dysfonctionnements à la fin sans que nous puissions trouver la raison, cependant avec la théorie que nous vous avons fourni, nous espérons que vous aurez compris le fonctionnement et ce que nous attendions.

Exécution :

« python gomokuGUI »
« python gomoku »