IA GOMOKU

1. INTRODUCTION

Dans nos premières réflexions pour avoir une IA Gomoku compétitive, nous avions pensé à un plan de base qui reposait sur : Une fonction fitness utilisant un réseau de neurones.

Pendant les premières implémentations, nous avons gardé l'idée du réseau de neurones couplé avec l'algorithme d'élagages alpha bêta. Nous l'avons entraîné contre une IA qui joue avec un algo alpha bêta pour qu'il prenne plus rapidement les bonnes décisions et qu'il calcule le moins de profondeur possible durant la partie (pour satisfaire les 5 secondes maximum).

Les différents problèmes que nous avons rencontrés sont souvent revenus avec la fonction fitness. Les premiers obstacles ont été sur le fait de faire une fitness plus orientée défense ou attaque. Nous l'avons ensuite appliqué au joueur qui jouait nous même. Par la suite, nous l'avons entraîné sur le réseau neuronal pour l'entraîner à prendre les meilleures décisions de défense ou attaque en fonction du résultat de l'algorithme alpha bêta, mais nous avons eu un problème pour calculer la fitness de chaque joueur indépendamment de son rôle dans la partie pour avoir les meilleurs résultats du minimax alpha bêta.

Au niveau du réseau de neurone, nous l'avons entraîné pendant 2 jours, le temps d'amasser les données et puis d'entraîner notre IA. Malgré ça, cela n'a pas été concluant, cf l'explication en dernière page.

2. STRUCTURE

Nous avons créé des classes joueurs pour les faire s'affronter, pour tester nos différentes fonctions fitness et les différentes modifications apportées au cours du projet.

Nous avons créé une matrice représentant le board du Gomoku pour que ce soit plus lisible et voir directement si les coups joués étaient mauvais.

Nous avons créé les modes blanc ou noir (premier à jouer ou second) ainsi que la règle que le 3e coups devait être forcément joué en dehors du carré principal.

```
class Player:
    def __init__(self,playerId,depth):
        self.depth = depth
        self.playerId = playerId
    def predict():
        return([-1,-1])
```

3. FONCTION FITNESS

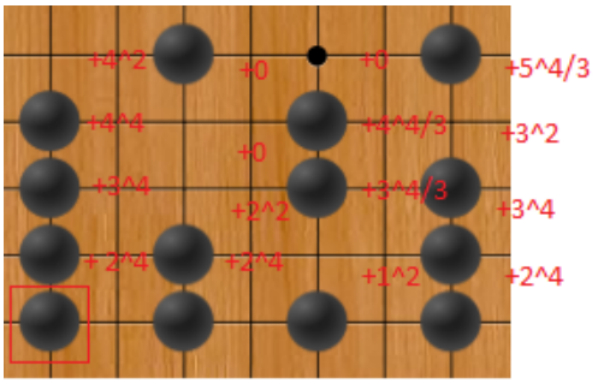
```
def fitness(x,player):
  c = 0
  s1 = 0
  s2 = 0
  for i in range15:
    for j in range15:
      vvx = x[i][j]
      if vvx == player:
        c += 1
        for w in vectors:
          wx,wy = w
          if (0 <= i+4*wx < 15 and 0 <= j+4*wy <
              15):
            tempS = 0
            prof = 0
            for k in range5:
              vx = x[i+k*wx][j+k*wy]
              if(vx == 0):
                prof += 1
                if(prof == 1): tempS+= k*k
              elif(vx == player):
                k2 = k+1 if k != 1 else 2.5
                tempS+= k2*k2*k2*k*(1 if prof ==
                    0 else (0.3 if prof == 1 else
                     0))
              else:
                tempS = 0
                break
            s1 += tempS
      elif vvx == -player:
        c += 1
        for w in vectors:
          wx,wy = w
          if (0 <= i+4*wx < 15 and 0 <= j+4*wy <
              15):
            tempS = 0
            prof = 0
            for k in range5:
              vx = x[i+k*wx][j+k*wy]
              if(vx == 0):
                prof += 1
                if(prof == 1): tempS+= k*k
              elif(vx == -player):
                k2 = k+1 if k != 1 else 2.5
                tempS+= k2*k2*k2*k2*(1 if prof ==
                     0 else (0.3 if prof == 1
                    else 0))
              else:
                tempS = 0
                break
            s2 += tempS
  fit = (s1-1.5*s2)/c
  return fit
```

Nous avons choisi de baser notre fonction fitness sur les différents pions alignés, cependant il faut qu'elle soit la plus optimisée possible et équilibrée, car le Gomoku est un jeu complexe avec beaucoup de structures. Pour chaque pion, on regarde dans toutes les directions et on incrémente nos points suivant ce que l'on rencontre et le nombre de pas fait dans cette direction.

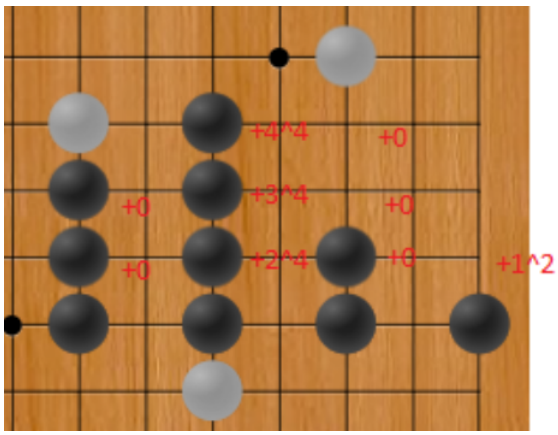
Si on rencontre la fin du plateau ou un pion adverse, les points associé à ce sens sont annulées, et si on rencontre une zone sans pion, on lui attribue quelques point et on le laisse regarder plus loin, toutefois il ne prend des point pour les pions de son équipe que si et seulement si l'espace est de maximum de 1-2 espaces même avec un pion entre

les deux ne donneront plus de points au-delà, mais si on rencontre un pion adverse les points sont tout de même annulés peu importe le nombre d'espaces.

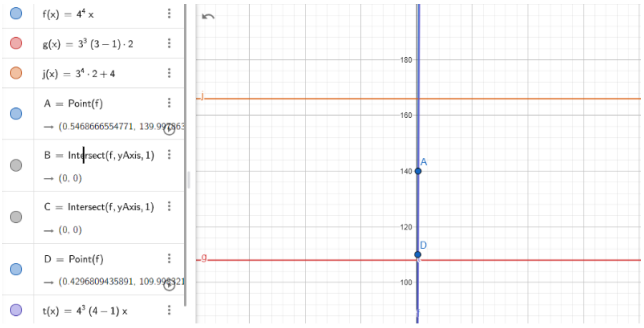
La partie la plus complexe est de bien équilibrer les différents patterns pour que l'IA ne choisisse pas un coup qui lui donne un avantage en passant à côté d'une ligne de 3 ou 4. Pour finir on calcule la différence entre nos points et ceux de l'adversaire multiplié par 1.5 car cette fonction n'est appelée qu'après un coup donc l'adversaire joue toujours après nous donc il faut bien que cet avantage se voit sur le score. Ceci donne lieu à des fitness non symétriques et majoritairement négatives mais cela n'as pas d'importance pour l'algorithm minmax



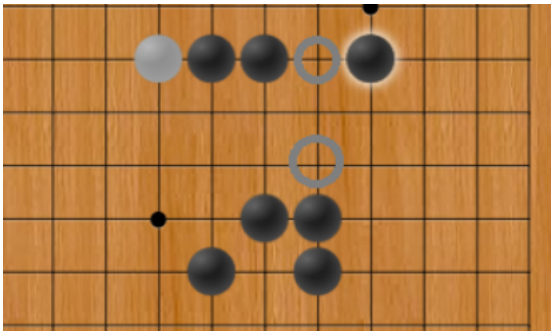
Point de l'adversaire



Nous avons choisi ces paramètres pour que l'IA puisse faire la différence entre une situation qui peut poser un problème et une situation qui peut causer une défaite au prochain coup. Il nous a fallu étudier certaines fonctions simples pour trouver des paramètres qui satisfassent toutes nos conditions.

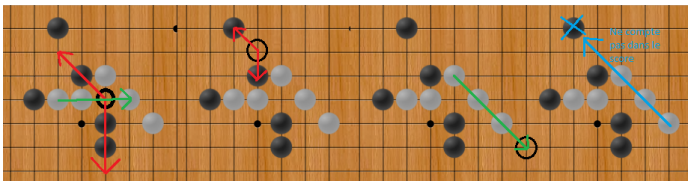


Nous avons rencontré par exemple ce problème :

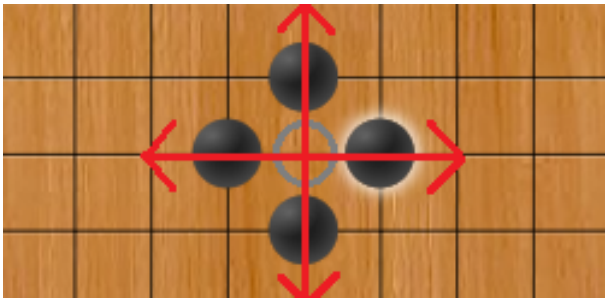


Ou l'IA choisissait de bloquer la situation du haut plutôt que celle du bas alors que la situation du haut demande 2 coups pour une victoire alors que celle du bas n'en nécessite qu'un.

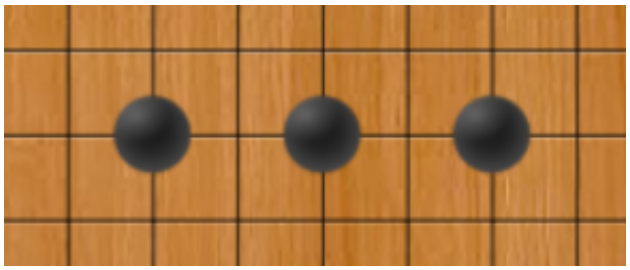
Dans ce cas simple, l'algorithme minimax suffit à résoudre le problème, mais quand cette structure se retrouve dans une partie, il est arrivé plusieurs fois à l'IA de jouer la mauvaise défense et donc nous avons réajusté nos paramètres. De même pour cette situation :



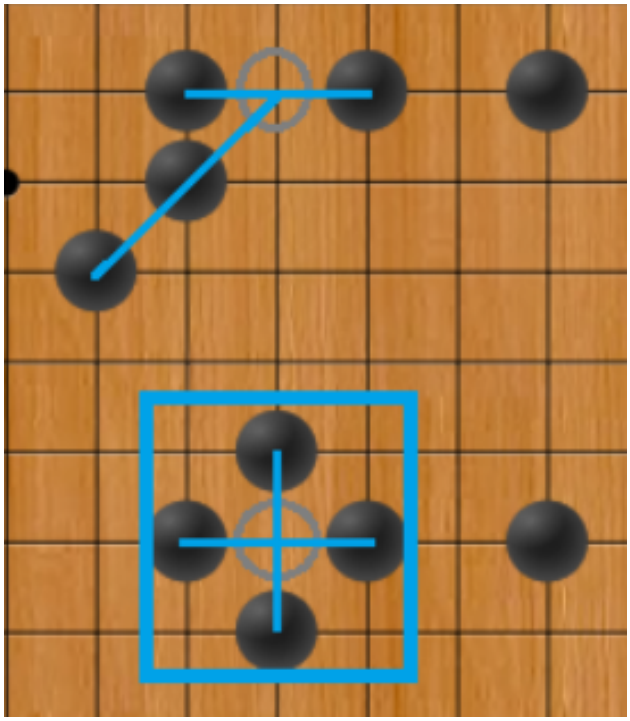
L'ajout de point pour les espaces est nécessaire sinon l'IA n'est pas capable de reconnaître ce genre de menaces.



Cependant, comme nous limitons les points à un seul espace, l'IA n'est pas capables de reconnaître totalement cette structure.



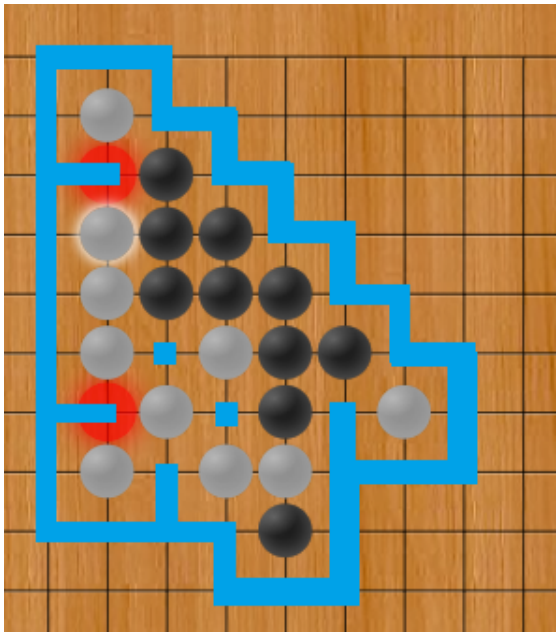
Cette structure ne permet par une victoire en un coup, elle repose donc sur d'autres structures déjà connues par l'IA



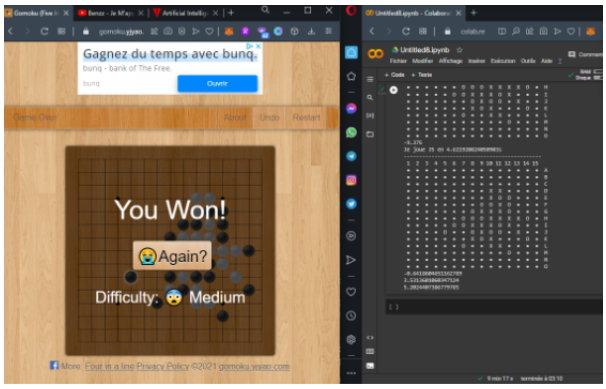
Si nous augmentons la limite d'espace alors l'IA var favoriser des coups éloigner pour maximiser ses points par coup et donc var jouer avec des pions écarté et vite perdre son avantage

Après avoir profilé nos fonctions, on a remarqué que 80% du temps de calcul était utilisé pour des calculs de fitness, nous avons ainsi cherché à optimiser la fonction au maximum, nous avons alors fait énormément de test sur les fonctions présentes en python et sur leurs différentes utilisations pour trouver quel est le plus rapide. On a par exemple stocké les range(15) dans une variable, car on cherche à avoir cette valeur très souvent.

Pour pouvoir atteindre le seuil des 5s sans perdre en qualité, nous avons implanté des paliers de profondeur avec une profondeur de maximum 4. Lorsque la recherche atteint 4.3 secondes, on arrête de calculer des nouvelles fitness de profondeur ¿ 2 puis lorsque la recherche atteint 4.9 secondes, on arrête de calculer de nouvelles fitness pour tous les nœuds et on prend le max de ce qu'on a entre les mains. Comme les coups possibles sont triés de haut en bas, l'IA a tendance à mieux connaître les coups en haut du plateau, lorsqu'elle joue contre elle-même les coups sont souvent orientés vers le haut. L'implémentation de minimax n'as rien de particulier et utilise uniquement les coups sur la première couche extérieure.



Et c'est ainsi que nous avons réussi à battre l'IA médium du site exemple (avec une fitness légèrement différente de celle utilisée pour les matchs)



4. RÉSEAU DE NEURONE

Notre première idée était d'utiliser un réseau de neurones que nous aurions entraîné sur des exemples donnés par notre fitness et par la suite fait jouer contre lui-même en l'entraînant sur les coups de ses parties suivant l'issue du match. Afin d'avoir une fonction fitness qui ne nécessite que des multiplications de matrice et donc très légère. Cependant, même si sur les données d'entraînement (environ 100000) le réseau avait une bonne précision, cependant sur de nouvelles situations la réponse n'avait aucun sens et par faute de temps, nous avons abandonné

cette idée et nous nous sommes concentrés sur ce que nous avions déjà fait en améliorant et optimisant notre fonction fitness. Vous pouvez retrouver dans le dossier nn.py qui est une classe réseau neuronal implanté sous python, les nn.txt les réseaux sauvegardés avec donnéesTrainExtrait un extrait des données utilisée pour l'entraînement. Nous avons utilisé une structure [225,150,80,10,1] (Input, hidden layers, output) et nous l'avons entraîné sur les données générées par les appels à la fonction fitness en prenant soin d'avoir un bon ratio de mauvais coup et de bon coup et beaucoup de situation simple.

Phase d'entraînement :