UTBM – Département Informatique

Projet – IA41

Semestre P2020

Sujet :

Teeko

Responsable :

Fabrice LAURI

Yassine RUICHEK

Huangkai ZHENG

Haoyun LIAO

Jiawei MAO

Hui TONG

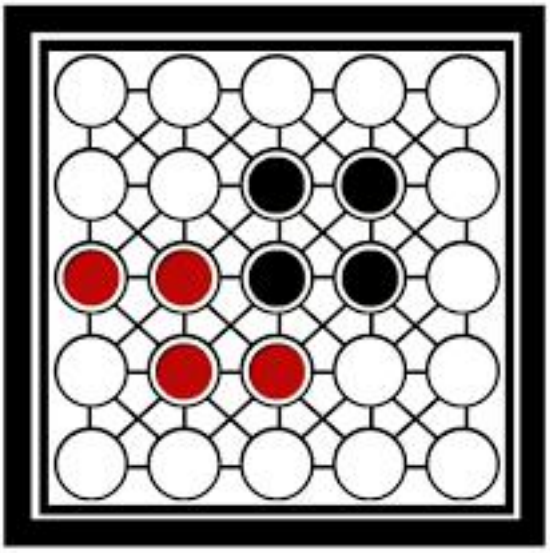
SOMMAIRE

Introduction

Dans l'UV IA41 nous sommes ammenés a réalisé un projet de programmation en Prolog afin d'utiliser les connaissances acquises dans l'unité de valeur ce semestre. Nous avons choisi le jeu de société Teeko. Dans ce rapport, nous allons expliquer les règles du jeu Teeko. Dans ce rapport, nous présenterons les règles du jeu Teeko. Nous parlerons également des problèmes que nous avons rencontrés dans le cadre du projet et de la manière dont nous les avons résolus. Plus important encore, nous expliquerons les principales parties du code.

Règles du jeu

Le jeu Teeko est un jeu pour deux joueurs, sur un plateau carré de 25 cases. Le but du jeu est d'aligner ses quatre jetons, que ce soit dans le sens horizontal, vertical ou encore diagonal, ou d'effectuer un carré de quatre cases adjacentes. Le joueur Noir débute la partie en plaçant son pion sur la case de son choix. C'est ensuite au joueur Rouge de placer son pion sur n'importe quelle autre case (la case doit être inoccupée), puis encore au joueur Noir et ainsi de suite jusqu'à ce que chacun des joueurs ait placé tous ses pions. Si aucun des joueurs n'a gagné après cette étape, chacun d'eux joue à son tour (le joueur Noir commence), ne déplaçant qu'un pion à la fois, et d'une case adjacente seulement.



1 Formalisation du problème

1.1 Utilisation de l'algorithme Min/max

L'algorithme Min/max, également connu sous le nom d'algorithme du maximum minimisé, est un algorithme qui trouve la plus petite des possibilités maximales d'échec. L'algorithme Min/max est couramment utilisé dans les jeux et les programmes tels que les échecs, où deux joueurs se relaient à chaque fois pour s'affronter. L'algorithme Min/max est la base des algorithmes de jeu basés sur la recherche. L'algorithme est un algorithme à somme nulle, dans lequel une partie doit choisir l'option qui maximise son avantage parmi les options disponibles, tandis que l'autre partie choisit l Une méthode pour minimiser l'avantage de l'adversaire.

L'algorithme Min/max va générer un arbre de jeu basé sur la forme actuelle. Les deux joueurs joueront à tour de rôle. Par exemple, si c'est à notre tour de jouer, cela va générer un nœud pour chaque coup possible. L'algorithme analysera ensuite les mouvements possibles de l'adversaire en fonction de ce nœud et générera un nœud pour chaque mouvement. Et ainsi de suite jusqu'à ce qu'un camp soit capable de gagner, ce qui génère un arbre de jeu.

La chose la plus importante pour l'algorithme est la profondeur de l'arbre et la valeur des nœuds. Si nous sommes le côté max (c'est-à-dire que le côté avec le score le plus élevé est choisi), alors l'adversaire est le côté min (c'est-à-dire que le côté avec le score le plus bas est choisi) . Ainsi, dans l'arbre, nous nous retrouvons par défaut dans la pire situation (Min) que notre adversaire nous donne à tous. Et nous voulons choisir la meilleure (Max) parmi les pires situations. C'est ce que montre le schéma suivant.

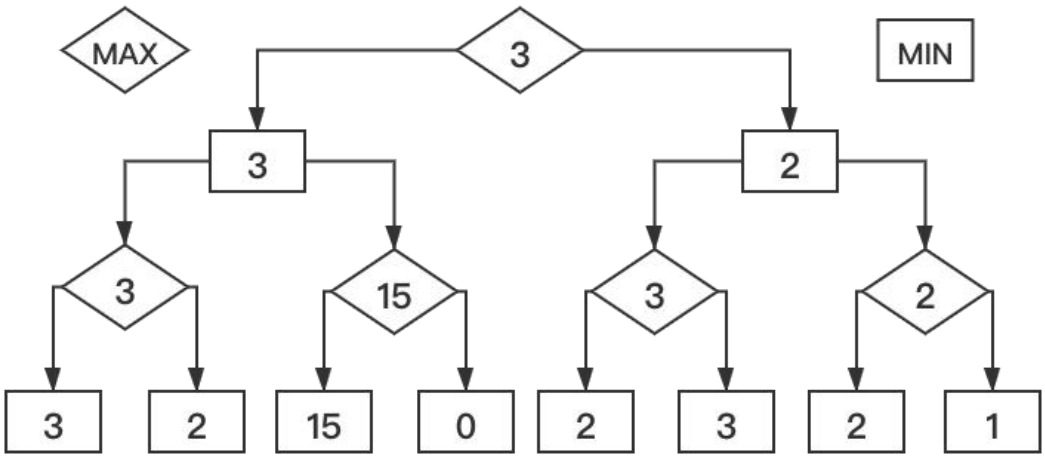


Illustration 1 :Exemple de l'algorithme Min/max

Pendant l'exécution de l'algorithme, nous devons définir la profondeur de l'arbre. La profondeur déterminera le nombre de couches de récursivité et la quantité d'intelligence acquise. Au fur et à mesure que la profondeur s'accroît, nous serons en mesure d'analyser la qualité de chaque choix. Le meilleur nœud enfant d'un nœud max est le plus grand de ses enfants nœuds min. à l'inverse, le meilleur enfant d'un nœud min est le plus petit de ses enfants nœuds max.

1.2 Utilisation d'alpha/bêta

Lorsque nous examinons le teeko en utilisant l'algorithme minmax, il existe de nombreuses possibilités dues à chaque branche. Le nombre de nœuds enfants de l'arbre augmente exponentiellement avec chaque couche de la branche. Prenons l'exemple du début de partie : le premier joueur à 24 possibilités, puis, après qu'il aie posé sont pion, son adversaire aura 23 possibilités pour poser le sien. (24 * 23 = 552) et ainsi de suite. Lorsque le second joueur aura joué 2 fois, on arrive déjà à 255024 dispositions de pion possibles sur le plateau (552 * 22 * 21 = 255024). Cela génère un nombre important de calcul, dont la grande majorité ne sont pas nécessaire! Nous avons donc besoin d'un algorithme pour soustraire les branches indésirables, ce qui réduira considérablement la quantité de calculs. Pour réduire davantage de nœuds enfants, nous avons choisi d'utiliser l'algorithme d'élagage alpha-bêta.

Chaque nœud aura une plage déterminée par les valeurs alpha et bêta [alpha, bêta], la valeur alpha représente la limite inférieure, et bêta représente la limite supérieure. Pour chaque nœud enfant recherché, la plage est corrigée selon les règles. Les nœuds max peuvent modifier la valeur alpha et les nœuds min modifient la valeur bêta. Si bêta <= alpha se produit, aucun sous-arbre n'est plus recherché, et la partie non recherchée du sous-arbre est ignorée, cette opération est appelée élagage(pruning).

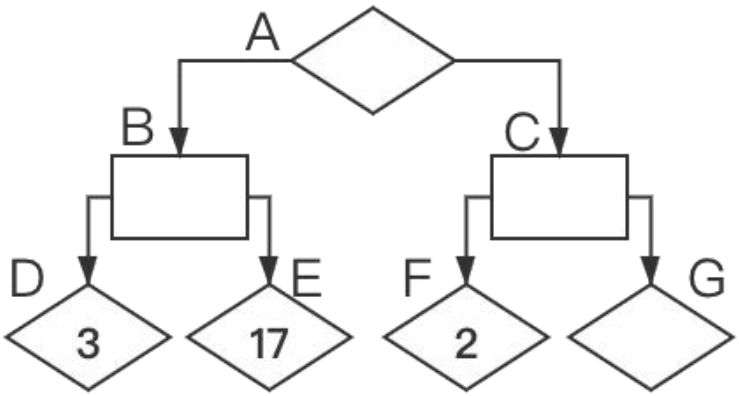


Illustration 2 :Exemple de l'algorithme alpha/bêta

En partant du nœud B, B est un nœud min et doit trouver le plus petit de D et E, donc B prend une valeur de 3 et le bêta de B est également fixé à 3. Supposons que B ait plus d'enfants que 3, mais comme la valeur minimale de D est déjà présente, elle n'a aucun effet sur B pour avoir un impact. Autrement dit, ici, bêta = 3 définit une limite supérieure.

A est le nœud maximum et vous devez trouver la valeur la plus élevée dans B et C. Puisque le sous-arbre B a été recherché et que la valeur de B est déterminée comme étant 3, la valeur de A est au moins 3, ce qui détermine l'alpha inférieur de A = 3. Avant de rechercher le sous-arbre C, nous voulons que C soit supérieur à 3 afin d'avoir un effet sur l'alpha inférieur de A. Ainsi, C obtient la limite inférieure alpha = 3 de A, ce qui est une restriction.

C est le nœud min, et il faut trouver le plus petit de F et G. Puisque la valeur de F est de 2. C doit donc être inférieur ou égal à 2. À ce stade, nous actualisons la limite supérieure de C pour qu'elle soit de bêta = 2. À ce stade, C a alpha = 3, bêta = 2, qui est un intervalle nul, ce qui signifie que même si nous continuons à considérer les autres enfants de C, il est impossible d'avoir une valeur de C supérieure à 3, donc nous n'avons pas besoin de considérer le nœud G. Le nœud G est le nœud qui est élagué.

En répétant le processus ci-dessus, un plus grand nombre de nœuds seront ignorés en raison des opérations d'élagage. Cela nous permet d'optimiser l'algorithme Min/max.

1.3. Analyser du Problème

Nous nous fixons comme objectif d'implémenter le jeu du Teeko permettant de jouer une partie humain contre ordinateur ou ordinateur contre ordinateur.

Pour pouvoir réaliser un jeu complet. Nous avons divisé le programme en plusieurs parties:

1. Définir le contenu du plateau (par exemple, 0 pour aucun pion, 1 pour les joueurs, 2 pour l'IA) et l'algorithme de déplacement des pions 2. écrire l'interface du jeu et interagir avec elle.

2. Ecrire des interfaces de jeu et interagir avec la capacité de le faire

3. Ecrire un algorithme de notation pour évaluer chaque mouvement possible et finalement arriver au meilleur point possible pour faire atterrir une pion.

2 Contrôler les pions

Dans cette partie, nous allons montrer comment nous contrôlons le placement et le mouvement des pions en arrière-plan.

2.1. Stock des pions

L'idée est d'utiliser un tableau à deux dimensions pour stocker des pions. Le tableau est rempli par 0 à initial, qui représentent il n'y a pas de pions dans le tableau. Ensuite, on utilise « 1 » pour représenter l'homme, « 2 » pour représenter l'ordinateur. Après chaque action, nous modifierons la valeur du tableau. Ensuite, l'interface graphique dessine des images basées sur notre tableau.

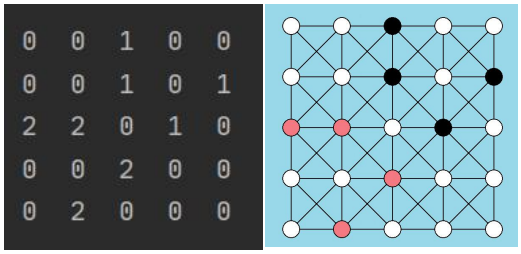


Illustration 3 : Exemple de tableau

Dans le même temps, nous avons un tableau pour enregistrer les positions des pions du joueur ou des pions d'ordinateur. Nous attribuons des valeurs à chaque position du tableau dans l'ordre, allant de 1 à 25. Le but de cette étape est de faciliter l'algorithme pour calculer la valeur.

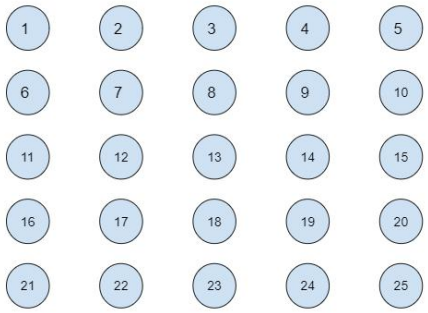


Illustration 4 : Exemple de position

2.2. Placement des pions

Nous avons une fonction putFlag pour contrôler les joueurs pour placer des pions. Dans le processus de placement, nous devons juger s'il y a un pion à cette position. Si non, le pion peut être placé dans cette position. Sinon, le joueur doit choisir une autre position pour placer le pion.

2.3. Détection les positions possibles à bouger

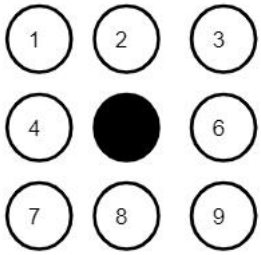


Illustration 5 : Le joueur peut choisir des positions

Pour chaque pion, nous avons numéroté 9 emplacements autour. Chaque fois que le joueur choisir une pion, nous allons scanner autour de la position où la pions peut être placée, ensuite la fonction retourne une tableau qui contient tous les positions possibles.

2.4. Déplacement les pions

Une fois que le joueur a placé 4 pions, l'opération suivante consiste à déplacer les pions pour gagner. Pour les déplacements, nous avons écrit la fonction changePosition. Il accepte une direction représentée par un entier et une coordonnée représentant une pion d'échecs. Lorsque nous aurons les coordonnées, nous utiliserons la méthode décrite dans la troisième étape pour obtenir la position où il peut se déplacer. Ensuite, nous nous déplaçons dans une certaine direction en fonction de l'entier entré.

```
def changePositionHelper(self,p,x,y):
    lastIndex=x*5+y+1
    if p==1:
        self.changeState(lastIndex,x-1,y-1)
    elif p==2:
        self.changeState(lastIndex,x-1,y)
    elif p==3:
        self.changeState(lastIndex,x-1,y+1)
    elif p==4:
        self.changeState(lastIndex,x,y-1)
    elif p==6:
        self.changeState(lastIndex,x,y+1)
    elif p==7:
        self.changeState(lastIndex,x+1,y-1)
    elif p==8:
        self.changeState(lastIndex,x+1,y)
    elif p==9:
        self.changeState(lastIndex,x+1,y+1)
```

Illustration 6 : Code de changePositionHelper()

2.5. Gagner le jeu

Chaque fois que le joueur ou l'ordinateur bouge, nous devons juger de la victoire ou de la perte du jeu. Nous avons écrit une fonction « whether_win » et une fonction « winhelper ». L'idée est la suivante :

Parcourez le tableau à chaque fois et arrêtez lorsqu'il rencontre la première valeur différente de 0. Ensuite, jugez selon les 4 situations de victoire.

```
#judge whether the player win
def whether_win(self):
    for i in range(5):
        for j in range(5):
            if self.coordinate[i][j] == self.turn:
                x=i
                y=j
                break
        else:
            continue
        break
    return self.winHelper(x, y)
```

Illustration 7 : Code de whether_win()

Carré :

Lorsque les valeurs droite, inférieure et inférieure droite de ce point sont toutes égales, il est jugé comme une victoire.

```
if x < 4 and y < 4 and self.coordinate[x][y + 1] == self.coordinate[x + 1][y] == self.coordinate[x + 1][y + 1] == self.turn:  # carré
    return True
```

Ligne :

Lorsque les valeurs des trois positions à droite du point de changement sont égales, cela est jugé comme une victoire.

```
elif y < 2 and self.coordinate[x][y + 1] == self.coordinate[x][y + 2] == self.coordinate[x][y + 3] == self.turn:  # ligne
    return True
```

Colonne :

Lorsque les valeurs aux trois positions en dessous du point lui sont égales, il est jugé comme une victoire.

```
elif x< 2 and self.coordinate[x + 1][y] == self.coordinate[x + 2][y] == self.coordinate[x + 3][y] == self.turn:  # colonne
    return True
```

Diagonale :

Dans les deux cas, la barre oblique gagne. En se penchant vers la droite, nous devons déterminer si les valeurs des trois positions obliques sont égales à la position actuelle.

En se penchant vers la gauche, nous devons déterminer si les valeurs des trois positions obliques sont égales à la position actuelle.

```
elif x < 2 and y < 2 and self.coordinate[x + 1][y + 1] == self.coordinate[x + 2][y + 2] == self.coordinate[x + 3][y + 3] == self.turn:  # diagonale
    return True
elif x < 2 and y > 2 and self.coordinate[x + 1][y - 1] == self.coordinate[x + 2][y - 2] == self.coordinate[x + 3][y - 3] == self.turn:  # diagonale
    return True
else:
    return False
```

2.6. Presque gagner le jeu

Pour faciliter le calcul de l'IA, nous avons également écrit une fonction pour déterminer si le joueur gagnera le jeu. La mise en œuvre spécifique est similaire à la partie 5. C'est juste que nous devons juger 3 positions au lieu de 4.

```
def nearlyWinHelper(self,x,y):
    if x < 4 and y < 4 and self.coordinate[x][y + 1] == self.coordinate[x + 1][y] == self.turn:  # carré
        return True
    elif x < 4 and y < 4 and self.coordinate[x][y + 1] == self.coordinate[x + 1][y + 1] == self.turn:  # carré
        return True
    elif x < 4 and y < 4 and self.coordinate[x + 1][y] == self.coordinate[x + 1][y + 1] == self.turn:  # carré
        return True
    elif x < 4 and y <= 4 and self.coordinate[x + 1][y - 1] == self.coordinate[x + 1][y] == self.turn:  # carré
        return True
    elif y <= 2 and self.coordinate[x][y + 1] == self.coordinate[x][y + 2] == self.turn:  # ligne
        return True
    elif x <= 2 and self.coordinate[x + 1][y] == self.coordinate[x + 2][y] == self.turn:  # colonne
        return True
    elif x <= 2 and y <= 2 and self.coordinate[x + 1][y + 1] == self.coordinate[x + 2][y + 2] == self.turn:  # diagonale
        if (x == 2 and y == 0) or (x == 0 and y == 2):
            return False
        else:
            return True
    elif x <= 2 and y >= 2 and self.coordinate[x + 1][y - 1] == self.coordinate[x + 2][y - 2] == self.turn:  # diagonale
        if (x == 0 and y == 2) or (x == 2 and y == 4):
            return False
        else:
            return True
    else:
        return False
```

Illustration 8 : Code de nearlyWinHelper()

3 Interface graphique

3.1. Page initiale

Nous utilisons PYQT5 comme interface visuelle pour réaliser l'interaction homme-machine. PyQt5 est une bibliothèque qui permet d'utiliser le framework Qt GUI de Python. Qt est un cadre de développement d'applications d'interface utilisateur graphique multiplateforme C ++ développé par Qt Company en 1991. Il peut développer des programmes GUI.

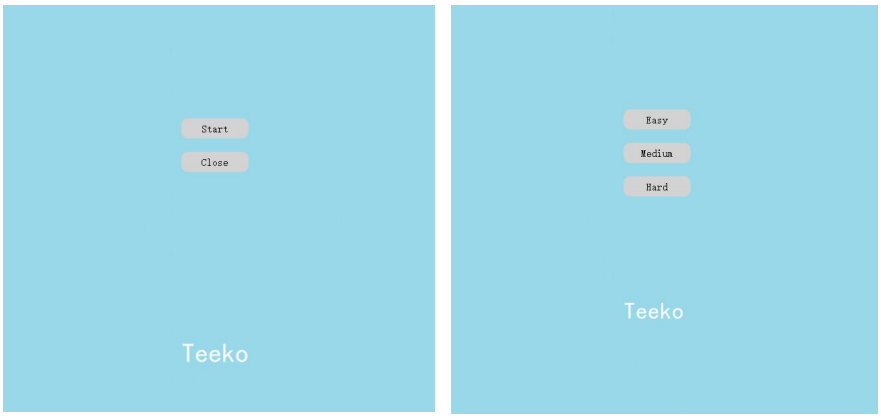


Illustration 9 : Exemple de page initiale

Une page de destination simple, et une page pour choisir la difficulté (chaque difficulté correspond à une profondeur différente)

La classe principale pour la visualisation est Window. Dans la classe Window, nous avons introduit quelques classes nécessaire de PYQT5 pour dessiner des échiquiers et des pions.

3.2. L'interface du jeu

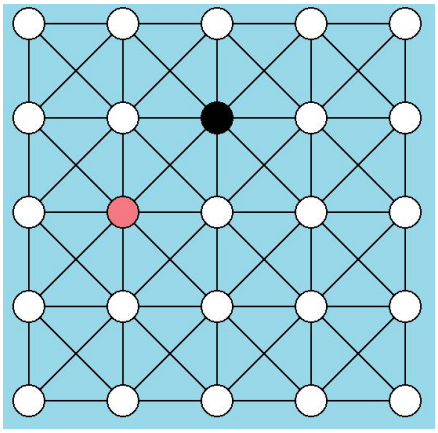


Illustration 10 : Exemple d'échiquier

Dans notre jeu, le noir représente le joueur et le rose représente l'ordinateur. Par défaut, le joueur joue en premier.

Afin d'obtenir une entrée utilisateur, j'ai réécrit mouseMoveEvent () et mousePressEvent () pour implémenter les événements de clic.

Dans mousePressEvent(), nous utiliserons les coordonnées réelles du curseur pour les convertir en coordonnées du tableau dans le jeu, ce qui signifie déterminer sur quelle pion d'échecs la souris clique.

La logique générale est qu'après que l'utilisateur a ajouté ou déplacé des pions, nous appelons des fonctions dans d'autres classes pour modifier le tableau de pions, puis la classe de fenêtre accepte à nouveau toutes les valeurs des paramètres et reflète toutes les modifications sur l'interface graphique.

Si le joueur déplace les pions incorrectement, il y aura un avertissement. L'interface graphique n'est qu'un outil, pas la partie centrale de ce jeu (algorithme Min-Max), mais nous voulons toujours l'améliorer et le rendre plus convivial. Nous avons rencontré des difficultés et les avons résolues

3.3. Problèmes rencontrés

Une difficulté est de savoir comment rendre le processus de déplacement des pions plus convivial après avoir joué quatre pions.

Au début, nous voulions utiliser le clavier pour saisir la direction du mouvement. La solution finale est de sélectionner une pion d'échecs avec le clic de la souris, et Sélectionner ensuite la grille à laquelle vous souhaitez déplacer, d'utiliser la différence de coordonnées pour obtenir la direction du mouvement et de modifier le tableau représentant les pions dans d'autres classes.

Nous n'avons pas pris en compte certaines situations, telles que l'utilisateur peut sélectionner la pion de l'adversaire, la grille vide peut être sélectionnée, etc. Plus tard, nous avons ajouté des conditions pour éliminer complètement ces erreurs supplémentaires.

Une autre difficulté est de savoir comment faire sentir à l'utilisateur que cette pion a été sélectionnée.

Après avoir cliqué sur la pion d'échecs, le clignotement de la pion informe l'utilisateur que la pion d'échecs est sélectionnée

Ceci est un morceau de code très intéressant. Nous lions le Qtimer à la fonction, donc à chaque fois (300ms), Qtimer appellera la fonction, en même temps, selon un index. Quand il est impair, nous dessinons la pions, quand il est pair, ne dessinez pas. Et chaque fois, nous appelons la fonction repaint(). Deux L'image alterne, créant un sentiment que les pions clignotent.

4 Implémentation de l'algorithme Min/max.

Dans cette partie, nous présenterons l'algorithme min max et notre méthode de calcul de valeur unique.

4.1. Introduction d'algorithme Min/max en python

Nous divisons l'algorithme en deux parties, la première partie est les huit premières étapes et la deuxième partie est le reste.

Parce que pour le teeko, les quatre premières étapes consistent à choisir un endroit libre pour jouer aux pions, et les autres étapes consistent à choisir et à déplacer vos propres pions. (Deux joueurs jouent quatre pions. Cela fait huit fois.)

```
if self.turn <8:    else:  # apres 8 fois
```

(Diviser en deux parties)

Nous utilisons "turn" pour enregistrer les étapes actuelles.

```
self.board.coordinate[y][x] = 2  #
tmp = self.max(self.depth - 1)  #
self.board.coordinate[y][x] = 0  #
if tmp < IAmin:  # 进行判断 如果局势变
    IAmin = tmp
    max_x = x
    max_y = y
```

(D'abord on le change en 2,après simulation on change en 0)

La manière spécifique est qu'après que les humains ont terminé ses partie, nous déplaçons n'importe lequel de nos poins dans n'importe quelle direction.

```
for y in range(5):
    for x in range(5):
        if self.board.coordinate[y][x] == 1:  # 对IA周围棋子进行遍历
            position = self.board.getPossiblePosition(y, x)
            for direction in position:
                ab=self.board.changePosition(direction, y, x)
                tmp = self.min(current_depth - 1)
                self.board.changeRole()
                self.board.changePosition(10-direction, ab[0], ab[1])  # 还原
                self.board.changeRole()
                if tmp > IAmax:
                    IAmax = tmp
```

Puis simulons le mouvement humain dans ces circonstances(dans max), et après avoir atteint une profondeur prédéterminée, selon qu'il s'agit de humains ou de pas robot, sélectionnez la pion avec la plus grande (ou la plus petite) valeur à déplacer.

Et, nous enregistrons ces mouvements.

Pour la valeur que nous fixons, plus la valeur est élevée, plus la probabilité de victoire humaine est élevée, nous choisissons donc min pour nos pas et max pour les pas humains.

4.2. Calculer la valeur

4.2.1. Pré-réflexion

La première chose qui me vient à l'esprit est que nous avons deux méthodes de calcul différentes selon qu'il s'agit d'un robot ou d'un humain.

Mais nous avons renversé cette idée, car la situation sur la carte est la même pour les humains et les robots.(L'un est positif et l'autre négatif).

4.2.2. Weights

```
self.weights = [
    [4, 6, 5, 6, 4],
    [6, 10, 10, 10, 6],
    [5, 10, 12, 10, 5],
    [6, 10, 10, 10, 6],
    [4, 6, 5, 6, 4]
]
```

Illustration 11 : Exemple de weights

Nous avons compté 44 plans gagnants et enregistré le nombre de fois que chaque position a été incluse comme weights de cette position

```
def playeraround(self, y, x):
    notrevalue = 0
    for i in range(10):
        if i != 5 and i != 0:
            position = self.board.getPossibleCheckPosition(y, x)
            for i in position:
                self.board.checkPosition(i, y, x)
                if self.board.coordinate[y][x] == 0:
                    notrevalue += int(self.weights[y][x]/2)
                elif self.board.coordinate[y][x] == 2:
                    notrevalue += 0
                else:
                    notrevalue += self.weights[y][x]
    return notrevalue
```

Nous avons écrit une fonction pour calculer le weight de chaque poin, si la position environnante est vide, ajoutez la moitié du weights de la position, si l'adversaire ,on ajoute pas, si c'est aussi sa propre poin, ajoutez le weight de cette position.

Bien sûr, ces valeurs sont positives pour les humains et négatives pour les robots.

4.2.3. La victoire

De plus, j'ai aussi considéré la situation de détection de victoire et la situation de victoire à venir.

Ces deux fonctions ont déjà été mentionnées dans la section précédente.

```
def whether_win(self):
```

```
def nearlyWin(self):
```

Chaque fois qu'une victoire est détectée, elle doit renvoyer un nombre avec une grande valeur absolue, car cela indique que la situation sur le terrain est déjà claire.Il en va de même pour la victoire à venir, ce qui signifie que nous approcherons de la victoire, et cela signifie que lorsqu'il est détecté que l'adversaire est sur le point de gagner, nous devons intercepter.

4.2.4 Alpha-beta

```
beta = self.max(current_depth - 1,alpha,beta)
self.board.changeRole()
self.board.changePosition(10-direction, ab[0], ab[1])
self.board.changeRole()
tmp = beta
if tmp < IAmin:
    IAmin = tmp
if alpha >= beta:
    return beta
```

Puisque le programme prend trop de temps à s'exécuter lorsque la profondeur est de 4. Nous avons donc choisi d'utiliser l'algorithme alpha/bêta pour réduire le nombre d'opérations.

On calcul max comme beta, min comme alpha,si alpha est plus que beta, les autres résultats de ce poins peut etre supprimé, donc on a améliorer la vitesse pour calculer.

5 Difficultés

Dans le cadre du projet, nous avons rencontré de nombreuses difficultés. Par exemple, comment définir l'intelligence de notre IA, comment créer les pages d'action, comment définir les propriétés dans le code, etc.

Comme notre code était écrit en plusieurs parties, nous avons passé beaucoup de temps à définir les méthodes et les variables lorsque nous avons finalement fait la somme du code. Par exemple, nous n'étions pas d'accord sur la définition de l'échiquier dans un programme qui peut déplacer des pions et dans un programme Min/max, ce qui nous a conduit à être en désaccord sur les définitions x et y de l'échiquier.Bien sûr, nous avons résolu le problème à la fin, mais cela nous a appris que nous devons nous mettre d'accord sur les variables et les interfaces dont nous avons besoin au tout début du projet pour faciliter le travail sur les projets ultérieurs.

Une autre difficulté réside dans l'accord sur les poids des planches. Comme le calcul des poids des nœuds peut grandement influencer le jugement du programme (IA), nous avons réfléchi à de très nombreuses façons de le faire. Au début, nous avons pensé que les lieux les plus centraux devraient être plus lourds, car plus l'espace autour d'eux est important. Mais est que cette définition ne nous a pas totalement convaincus. Nous avons donc réfléchi à une nouvelle solution et redéfini les poids.

Bien sûr, nous avons également essayé d'utiliser l'algorithme alpha/bêta, mais nous n'avons pas réussi à mener à bien le projet.

6 Conclusion

Dans le cadre du projet, nous avons étudié attentivement les règles du teeko et l'application de l'algorithme Min/max et avons écrit des programmes pour mettre en oeuvre l'algorithme Min/max. Afin de rendre les IA plus intelligentes, Nous avons ajusté beaucoup de détails après les tests. Par example nous avons conçu différents poids pour que valeur calculée par ordinateur est plus raisonnable. Nous avons également modifié des conditions de jugement pour mieux juger de la situation aux limites. Même si vous comprenez l'algorithme bien, il y a encore beaucoup de détails qui nécessitent de l'attention

L'intelligence artificielle est de plus en plus utilisée dans le monde d'aujourd'hui. Par example, AlphaGo, algorithmes de reconnaissance de visage. Ce projet n'est qu'un simple projet dans le domaine de l'intelligence artificielle. Nous espérons que l'expérience acquise dans ce projet apportera une nouvelle compréhension à notre futur apprentissage de l'intelligence artificielle.