PRAVINE CALIAPPANE BTS SIO SLAM A

Jeu D’Echec en Python

Le jeu d'échecs est un jeu de plateau stratégique et tactique pour deux joueurs. Chacun contrôle une armée composée de seize pièces : un roi, une reine, deux tours, deux cavaliers, deux fous et huit pions. Le but du jeu est de mettre en échec et mat le roi adverse, ce qui signifie que le roi est menacé de capture et ne peut pas éviter cette menace.

Composition d’un jeu d’échecs

On peut décomposer un logiciel d’échecs aux éléments suivants :

Un échiquier ;

des pièces d’échecs;

un moteur de jeu;

une interface graphique.

Ainsi, le projet que je vous présente est composé actuellement des fichiers :

main.py : le programme principal qui affiche l'échiquier en mode console et répond aux sollicitations de l'utilisateur ;

engine.py : le moteur du jeu, capable d'analyser une position de l'échiquier et déterminer le meilleur coup ;

board.py : comprend les attributs et les méthodes de l'objet 'échiquier';

piece.py : comprend les attributs et les méthodes des pièces d'échecs.

L'échiquier

Il existe plusieurs façons de représenter l'échiquier. Etant composé de 64 cases, le plus simple en programmation est d'utiliser un tableau de 64 éléments. Les index de ce tableau seront numérotés de 0 à 63. Ainsi la case a8 correspond à 0, et h1 à 63.

```
coord=[
    'a8','b8','c8','d8','e8','f8','g8','h8',
    'a7','b7','c7','d7','e7','f7','g7','h7',
    'a6','b6','c6','d6','e6','f6','g6','h6',
    'a5','b5','c5','d5','e5','f5','g5','h5',
    'a4','b4','c4','d4','e4','f4','g4','h4',
    'a3','b3','c3','d3','e3','f3','g3','h3',
    'a2','b2','c2','d2','e2','f2','g2','h2',
    'a1','b1','c1','d1','e1','f1','g1','h1',
    ]
```

Avec la définition du tableau ci dessus, coord[3] retournera 'd8'.

Sur les cases de l'échiquier on peut mettre une pièce… ou pas.
Donc pour des raisons pratiques, on mettra toujours un objet-pièce sur chaque case de l'échiquier mais cet objet pourra être vide :

```
self.cases = [
    Piece('TOUR','noir'),Piece('CAVALIER','noir'),Piece('FOU','noir'),Piece('DAME','noir'),Piece('ROI','noir'),Piece('FOU','noir'),Piece('CAVALIER','noir'),Piece('TOUR','noir'),
    Piece('PION','noir'),Piece('PION','noir'),Piece('PION','noir'),Piece('PION','noir'),Piece('PION','noir'),Piece('PION','noir'),Piece('PION','noir'),Piece('PION','noir'),
    Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),
    Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),
    Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),
    Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),Piece(),
    Piece('PION','blanc'),Piece('PION','blanc'),Piece('PION','blanc'),Piece('PION','blanc'),Piece('PION','blanc'),Piece('PION','blanc'),Piece('PION','blanc'),Piece('PION','blanc'),
    Piece('TOUR','blanc'),Piece('CAVALIER','blanc'),Piece('FOU','blanc'),Piece('DAME','blanc'),Piece('ROI','blanc'),Piece('FOU','blanc'),Piece('CAVALIER','blanc'),Piece('TOUR','blanc')
    ]
```

Les pièces

Comme vous le savez les pièces sont : roi, dame, tour, cavalier, fou, pion. Certaines sont plus importantes que d'autres. On peut donc leur attribuer une valeur, par exemple 9 points pour la dame, 5 pour la tour, etc. Comme le roi ne peut pas être pris, il n'est pas utile de lui donner une valeur.

```
# Nom des pieces
nomPiece=(VIDE,'ROI','DAME','TOUR','CAVALIER','FOU','PION')

# Attribuer une valeur de score à chaque pièce : ROI=0, DAME=9, TOUR=5...
valeurPiece=(0,0,9,5,3,3,1)
```

Avec le code ci-dessus, une case vide et le roi ont 0 point, la dame 9, la tour 5, cavalier et fou 3 et enfin le pion : 1 point.

Une pièce a les attributs suivants :

un nom (roi, dame, tour…);
une couleur (blanc, noir);
une valeur (définie en points ci-dessus).
On peut les déplacer d'une case à une autre selon les règles propres à chacune. On va donc créer les fonctions permettant de trouver les cases de destination d'une pièce d'après sa case de départ.

Déplacement des pièces
Prenons un cas concret :
On a une tour en case a4 (index 32) (voir image plus haut des cases numérotées).
On va la déplacer d'une case à gauche. Elle se retrouve donc en case 32-1=31 soit h5 ! impossible !

Nous utiliserons donc la méthode de Robert Hyatt appelée 'mailbox' qui permet d'éviter ce débordement. Il s'agit de deux tableaux de 64 et 120 éléments, ressemblant à une boite à lettres selon son créateur, d'où son nom :

```
tab120 = (
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1,  0,  1,  2,  3,  4,  5,  6,  7, -1,
-1,  8,  9, 10, 11, 12, 13, 14, 15, -1,
-1, 16, 17, 18, 19, 20, 21, 22, 23, -1,
-1, 24, 25, 26, 27, 28, 29, 30, 31, -1,
-1, 32, 33, 34, 35, 36, 37, 38, 39, -1,
-1, 40, 41, 42, 43, 44, 45, 46, 47, -1,
-1, 48, 49, 50, 51, 52, 53, 54, 55, -1,
-1, 56, 57, 58, 59, 60, 61, 62, 63, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1
)
tab64 = (
21, 22, 23, 24, 25, 26, 27, 28,
31, 32, 33, 34, 35, 36, 37, 38,
41, 42, 43, 44, 45, 46, 47, 48,
51, 52, 53, 54, 55, 56, 57, 58,
61, 62, 63, 64, 65, 66, 67, 68,
71, 72, 73, 74, 75, 76, 77, 78,
81, 82, 83, 84, 85, 86, 87, 88,
91, 92, 93, 94, 95, 96, 97, 98
)
```

Avec notre tour en a4 : dans le tableau 'tab64' à l'index 32, nous obtenons la valeur 61.
On la déplace d'une case à gauche, on obtient 60.
Regardons le 60ème élément dans le tableau 'tab120' : cela donne -1.
Il s'agit de la valeur définie pour indiquer une sortie de plateau.

Toujours d'après ce tableau 'tab64' on peut définir les 'vecteurs' suivants pour les déplacements des pièces :

```
deplacements_tour=(-10,10,-1,1)
deplacements_fou=(-11,-9,11,9)
deplacements_cavalier=(-12,-21,-19,-8,12,21,19,8)
```

Et cerise sur le gâteau ça suffit ! inutile de spécifier des vecteurs pour la dame et le roi car ceux-ci se déplacent à la fois comme le fou et la tour, mais d'une seule case à la fois pour le roi.

Intelligence artificielle : Le moteur de jeu
Nous entrons dans le cœur du sujet.
Inutile de réinventer la roue, des mathématiciens ont imaginé des algorithmes de recherche du meilleur coup à jouer. Le plus approprié pour les échecs est l'algorithme alpha-béta.

Concrètement, cette technique permet de créer un arbre de positions de l'échiquier en attribuant pour chacune une note d'évaluation.

L'évaluation
Chaque position de l'échiquier est analysée selon plusieurs critères. Une note est calculée en fonction des pièces présentes et de nombreux bonus/malus. Il s'agit de 'l'évaluation'.
Voici la plus simple des fonctions d'évaluation ne prenant en compte que le matériel présent :

```
def evaluer(self):

    """Une fonction d'évaluation
    ne renvoyant actuellement que le score matériel"""

    WhiteScore=0
    BlackScore=0

    # Analyse des cases de l'échiquier de 0 à 63.
    for pos1,piece in enumerate(self.cases):

        # Score matériel
        if(piece.couleur=='blanc'):
            WhiteScore+=piece.valeur
        else:
            # NB : ici pour les pièces noires ou les cases vides.
            BlackScore+=piece.valeur

    if(self.side2move=='blanc'):
        return WhiteScore-BlackScore
    else:
        return BlackScore-WhiteScore
```

On pourrait compléter cette fonction d'évaluation avec les bonus et malus suivants (et en plus, le malus des uns fait le bonus des autres) :

pion passé : pion qui ne rencontrera plus de pion adverse et qui peut donc aller en promotion;
tour en 7ème rangée;
roi dans une colonne ouverte;
pions du roque avancés;
le roi a été déplacé et ne peut plus roquer;
pions doublés (2 pions dans la même colonne);
pion isolé : pion qui ne peut plus être protégé par un autre pion.
On peut également accorder des bonus/malus selon l'avancement de la partie :

en début de partie, il est préférable de roquer pour protéger le roi. Au contraire, en fin de partie, il faut placer le roi au centre pour éviter un mat sur une arrête ou dans un coin de l'échiquier;
un cavalier peut atteindre un plus grand nombre de cases s'il est au centre;
les pions prennent de la valeur s'ils approchent la 8ème rangée.
La recherche du meilleur coup
C'est à l'ordinateur de jouer. Il génère la liste de tous les coups possibles. Il déplace une pièce. Il attribue une note à la position. Il se mets à la place de l'adversaire et il fait exactement la même chose, et ainsi de suite. Il faut donc préciser au moteur de recherche une profondeur fixe (par exemple 5 déplacements) ou un temps donné de réflexion. Le programme va donc élaborer un arbre de recherche. Il construit une arborescence de positions de l'échiquier où chacune est notée.

Le meilleur coup est enregistré dans un tableau nommé 'Triangular PV-Table' (Principal Variation). Il s'agit d'un tableau d'éléments indexés par le numéro de profondeur en rapport au début de la recherche. Il a pour but de collecter la meilleure ligne de coups successifs. Les éléments sont les meilleurs coups trouvés et enregistrés dans alpha-beta. Il est donc recommandé de suivre cette ligne de coups afin de permettre un élagage efficace, puis les coups avec prise.

Voici l'algorithme alphabéta :

```python
def alphabeta(self,depth,alpha,beta,b):
    # Nous sommes arrivés à la fin de la recherche : renvoyez le score du plateau

    if(depth==0):
        return b.evaluer()
        # TODO : return quiesce(alpha,beta)
    self.nodes+=1
    self.pv_length[b.ply] = b.ply
    # # Ne pas aller trop loin

    if(b.ply >= self.MAX_PLY-1):
        return b.evaluer()

    # Extensions
    # Si le roi est en échec, allons plus loin

    chk=b.in_check(b.side2move) # 'chk' used at the end of func too
    if(chk):
        depth+=1

    # TODO
    #trier les mouvements : d'abord les captures
    # Générer tous les mouvements pour le côté en mouvement. Ceux qui
    # laissent le roi en échec seront traités dans domove()
    mList=b.gen_moves_list()

    f=False # Drapeau pour savoir si au moins un mouvement sera effectué
    for i,m in enumerate(mList):

        # Effectuez le mouvement 'm'.
        # Si cela met le roi en échec, annulez-le et ignorez-le
        # Rappel : un mouvement est défini avec (pos1, pos2, promouvoir)
        # Par exemple : 'e7e8q' correspond à (12, 4, 'q')
        if(not b.domove(m[0],m[1],m[2])):
            continue

        f=True # Un mouvement a été réaliser
        score=-self.alphabeta(depth-1,-beta,-alpha,b)
        # Annuler le mouvement

        b.undomove()

        if(score>alpha):

            # TODO
            # Ce mouvement a provoqué une coupure,
            #il devrait être ordonné plus haut pour la prochaine recherche
            if(score>=beta):
                return beta
            alpha = score

            # # Mise à jour du tableau PV triangulaire
            self.pv[b.ply][b.ply] = m
            j = b.ply + 1
            while(j<self.pv_length[b.ply+1]):
                self.pv[b.ply][j] = self.pv[b.ply+1][j]
                self.pv_length[b.ply] = self.pv_length[b.ply + 1]
                j+=1
    # Si aucun mouvement n'a été effectué : c'est un NUL ou MAT
    if(not f):
        if(chk):
            return -self.INFINITY + b.ply # MAT
        else:
            return 0 # NUL

    # TODO
    # 50 mouvements règles

    return alpha
```

APERCU DU JEU