ECHECS

Deux versions sont proposées :

- la 1ère, en console, n'est pas jouable car il y a un risque de confusion permanent des pièces qui ne sont différenciées que par l'usage de lettres majuscules (blancs) ou minuscules (noirs). Cependant leurs mouvements semblent correctement implémentés, pour ce que j'ai pu en juger en jouant 5 minutes au jeu. Difficulté également de saisir au clavier le mouvement d'une pièce (la référence n'est pas évidente aux coordonnées, indiquées comme les pièces par lettres (colonnes) ou chiffres (lignes). Le besoin de cliquer d'abord sur la touche entrée n'est pas compréhensible et gène également la fluidité du jeu.
- Le 2de offre un aspect attractif car beau graphisme ; l'usage de la souris étant prévu, les cases sont reconnues mais c'est tout. Aucun développement au-delà : le roi noir juste se place dans la case cliquée et la console affiche ligne et colonne correcte.

Entre ces deux versions non achevées, on comprend que l'essentiel a été saisi du jeu. Le temps ayant manqué pour traduire le mouvement des pièces à l'aide des instructions du module tkinter.

Pour la version 1 qui fait 1300 lignes… des simplifications sont possibles, par exemple ce test :

```
if(c[0] not in letters):
        return err[1]
if(c[1] not in numbers):
        return err[1]
if(c[2] not in letters):
        return err[1]
if(c[3] not in numbers):
        return err[1]
```

pourrait être avantageusement remplacé par :

```
if not(c[0] in letters and c[1] in numbers and c[2] in letters and c[3] in numbers):
        return err[1]
```

Trois classes sont utilisées (Engine, Board et Piece) mais à quoi servent-elles ? Les fonctions et variables de classe ne sont pas bien claires non plus : à quoi sert self.pv, une liste à deux indices initialisée par la fonction clear_pv : self.pv=[[0 for x in range(self.MAX_PLY)] for x in range(self.MAX_PLY)]
Il manque des commentaires pour ces points. On dirait que ces fonctions dénotent un mécanisme complexe qui n'a pas été complètement compris, juste recopié du modèle trouvé sur internet…
Par exemple, la fonction isEmpty de la classe Piece :

```
def isEmpty(self):
        """Retourne vrai ou faux à la question "Il y a une pièce sur la case ou pas ?" """
        return (self.nom==self.VIDE)
```

Elle ne retourne pas True si il y a une pièce sur la case comme l'indique le commentaire, mais au contraire, elle renvoie True si le nom de la piece en cours est vide (s'il n'y a pas de pièce).
Dans la même classe, le commentaire « Mouvements du château » est trompeur, il s'agit du roque.

D'une façon générale, j'aurai préféré que vous assumiez mieux le fait d'avoir repris le code d'un programme d'échec qui fonctionne, en mentionnant le gros effort de compréhension que vous avez fait pour tenter de le comprendre, au moins partiellement. Les commentaires, je suppose, dénotent cet effort. Il me semble par ailleurs que la traduction n'est pas achevée, vu qu'on retrouve beaucoup de notations anglaises des pièces, ce qui, à priori, ne devrait pas arriver puisqu'on utilise une notation française.
En voyant ce bloc d'instructions qui défini le jeu :

```
c=input('>>> ')
if(c=='quit' or c=='exit'):  exit(0)
elif(c=='undomove'):       e.undomove(b)
elif('setboard' in c):         e.setboard(b,c)
```

```
elif(c=='getboard'):       e.getboard(b)
elif(c=='go'):             e.search(b)
elif(c=='new'):            e.newgame(b)
elif(c=='bench'):          e.bench(b)
elif('sd ' in c):          e.setDepth(c)
elif('perft ' in c):       e.perft(c,b)
elif(c=='legalmoves'):     e.legalmoves(b)
else:                      e.usermove(b,c) # coup à jouer ? ex : e2e4
```

Je suis un peu stupéfait car je comprends qu'il y a beaucoup d'autres interactions qui sont prévues, sans doute le jeu qui était développé dans ce programme avait plusieurs boutons de réglage qui permettaient de faire toutes sortes d'action (rejouer un coup, bénéficier d'une aide, réinitialiser, etc.) Vous auriez dû, à mon avis, simplifier beaucoup plus votre programme en supprimant tout ce qui n'est pas directement le mouvement d'une pièce.

Votre travail, dans ce contexte, aurait pu se concentrer sur la traduction graphique : il faut déterminer la ligne et la colonne saisie par un premier clic, traduire cela dans la codification du 1er programme (par exemple ligne=2, colonne=3 se traduit par 'c2'), faire de même pour un 2ème clic, et lancer le programme de vérification. Cela aurait été plus satisfaisant pour vous d'obtenir le rendu graphique final avec un dynamisme correct des pièces ; les histoires de prise en passant, de roque, de rejouer une pièce, etc. ne sont que des améliorations futures éventuelles.

Il est difficile de lier le second programme au 1er car l'approche est différente. Ce doit être faisable en l'intégrant directement dans le 1er et en modifiant ce qui doit être modifié : une sorte de module de saisie doit être fabriqué qui attend deux clics de souris : au 1er on vérifie si la case contient bien une pièce de la couleur du tour, au 2d clic, on vérifie que le mouvement est possible et on l'exécute. Pour effectuer cette jonction, il serait peut-être judicieux de n'importer que ce qui est strictement nécessaire, quitte à en réécrire certaines parties.

Je trouve que ce 2d programme est ce qu'il y a de plus réussi dans ce projet : c'est simple, compréhensible par tous et le rendu graphique est excellent. Cela aurait été logique d'y incorporer progressivement en les adaptant, les différents éléments utiles du 1er programme.

J'ai écrit une version plus avancée de ce programme graphique (1_INTERFACE_GRAPHIQUE_2.py) que vous pourrez compléter si vous le souhaitez. Cette version réalise le déplacement des pièces et la prise d'une pièce ennemie avec affichage des pièces prises de part et d'autre de l'échiquier ainsi qu'un affichage formaté en console pour retrouver l'historique des coups. Le mouvement légal des pièces n'a pas été déployé (j'ai commencé en écrivant ce qui règle le mouvement des pions blancs, cela reste à généraliser aux pions noirs d'abord, puis aux autres pièces) car il est plus laborieux à réaliser. Je me suis empêché d'utiliser les classes de l'autre programme pour ne pas avoir à faire le ménage (opération longue et pour le coup très laborieuse) et j'ai voulu conserver le squelette du 2ème programme pour montrer qu'un programme peut évoluer dans le bons sens, même s'il n'a pas été correctement pensé au départ. Ce programme, dans l'état, permet de jouer comme si on avait un jeu réel (les règles n'étant pas connues, les pièces bougent alternativement et peuvent aller n'importe où sur l'échiquier, même prendre le roi adverse…) excepté du roque qui n'est pas possible (une seule pièce est déplacée à chaque fois), de la prise en passant et de la promotion d'un pion en reine.

Projet de note : 15/20
Détail de la note :

- préprojet : 1.5/2 (le projet est un peu trop vaste ; beaucoup de points sont vagues car leur réalisation n'est pas cernée, l'affichage notamment)

- compte-rendu : 3.5/4 (bonne narration de la difficulté rencontrée pour lier l'affichage graphique et de la solution adoptée, on pourrait adjoindre une illustration au moins du rendu obtenu)
- Programme : 10/14 (sans doute fonctionnel pour le programme console mais pas utilisable dans la pratique car pas de couleur ; partie graphique inachevée et trop éloignée de la partie fonctionnelle)