UFR de mathématique et d'informatique
Université de Strasbourg

Algorithmique et Programmation 1 : TP Noté Automne 2021

Durée : 1h30 ; Tous documents autorisés; Aucune communication autorisée.

Consignes :

Dans ce TP, chaque question vous invite à concevoir et écrire une fonction en python.

1. Ecrire tout votre code **dans un fichier** avec exactement le nom `tp.py`. Vous le déposerez sur Moodle à la fin. Notez que vous pouvez déposer des versions intermédiaires qu'il est possible de remplacer jusqu'à la cloture du dépôt.
2. Utiliser au maximum les fonctions écrites dans les questions précédentes.
3. Chacune de ces questions est suivie d'une rubrique appelée *Tests* qui spécifie les valeurs et comportements attendus pour différentes valeurs des arguments. Nous vous invitons à tester vos fonctions au moins avec ces appels. Une fois vos fonctions testées, nous vous demandons de ne pas les supprimer mais de les laisser sous forme de commentaire.

Les deux fonctions que l'on vous donne (`init` et `affiche` sont téléchargeables sur la page Moodle du cours).

1. Contexte

Au jeu d'échecs, une Reine se déplace et peut capturer les pièces à toute distance en ligne, colonne ou diagonale. Dans cet exercice, on vérifiera si plusieurs reines placées sur l'échiquier peuvent se capturer.

L'échiquier (8x8 cases) est représenté par une liste de listes, qu'on appelera aussi matrice dans la suite. On vous donne ci-dessous, une fonction d'initialisation d'un échiquier vide : nous avons choisi de coder une case vide par une valeur 0.

```
def init() :
  return [   [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0],
             [0,0,0,0,0,0,0,0]
            ]
```

On vous donne ci-dessous, une fonction permettant d'afficher l'échiquier. Intégrez cette fonction dans votre code pour des vérifications visuelles ultérieures.

```
def affiche( e ):
  larg_case = 3
  trait = '-' * (8*larg_case+9)
  case_vide = ' ' * larg_case
  nrows = len(e)
  ncols = len(e[0])
```

```
  for lin in range(nrows):
    print('  ' + trait)
    # affiche le numero de ligne
    print(f'{8-lin} ', end='')
    # affiche les cases de la ligne
    for col in range(ncols):
      if e[lin][col] == 0:
        print(f'|{case_vide}', end='')
      else:
        print(f'| {e[lin][col]} ', end='')
    print('|')
  # dernier trait horizontal
  print('  ' + trait)
  # affiche nom colonne
  print('   ', end = '')
  for lettre in range(ncols):
    nom = chr(ord('A') + lettre)
    fill = ' '
    print(f"{fill}{nom}{fill} ", end = '')
  print('')
```

2. Manipulation de l'échiquier (4 pts)

Placer une pièce (2pt)

Ecrire une fonction `place_piece` qui prend en paramètres :

- `e` : une liste de listes représentant l'échiquier,
- `coord` : un couple d'entiers (tuple) représentant l'élément de la liste de listes qu'on souhaite modifier,
- `piece` : une chaine de caractères qui représente la pièce.

Cette fonction retourne `True` si la liste de listes a été modifiée, `False` sinon.

Pour une coordonnée, on appelle *ligne* le premier élément du couple, *colonne* le deuxième. La ligne correspond au numéro de sous-liste dans la liste de listes, la colonne au numéro d'élément dans cette sous-liste.

Cette fonction place simplement `piece` aux coordonnées indiquées, après avoir vérifié la validité des coordonnées. Si les entiers dans `coord` ne sont pas compris entre 0 et 7, la fonction retourne `False` sans modifier `e`. Sinon, elle modifie l'élément de `e` et rend `True`.

Tests

```
echiquier = init()
result =  place_piece(echiquier, (1, 0), 'Q')
if result:
  print(echiquier)
else:
  print("coordonnées invalides")`
```

doit afficher un `Q` au début de la deuxième sous-liste.

```
[[0, 0, 0, 0, 0, 0, 0, 0], ['Q', 0, 0, 0, 0, 0, 0, 0], ... ]
```

Convertir une notation algébrique en coordonnées (2pt)

Aux échecs, on désigne une case par un nom de colonne (A, B, ..., H) et numéro de ligne (1, 2, ..., 8) (dans cet ordre, donc dans l'ordre inverse des lignes à celui utilisé précédemment et en commençant à 1). Dans la suite on appelle ce système de numérotation la *notation algébrique*.

Etant donné la représentation habituelle du jeu d'échecs, avec les joueurs blancs en bas, on a la correspondance suivante :

- les colonnes sont notées A, ..., H. La lettre 'A' correspond à la colonne 0, 'H' à la colonne 7 dans la liste de listes.
- les rangées sont notées 1, ... , 8. La rangée 1 correspond à la ligne 7, la rangée 8 correspond à la ligne 0.

Par exemple, la case `"B5"` correspond aux coordonnées `(3, 1)`.

Ecrire une fonction `convert_vers_coord` qui prend en paramètre une notation algébrique, c'est-à-dire une chaîne de deux caractères formée d'une lettre (A-H) et d'un entier (1-8), et renvoie la coordonnée de la case dans la matrice. On suppose que la notation algébrique est correcte.

Indication: vous pouvez utiliser la fonction `ord()` pour passer de la lettre à l'entier correspondant.

Tests

```
print(convert_vers_coord("B5"))   # -> (3, 1)
print(convert_vers_coord("H1"))   # -> (7, 7)
```

3. Mouvement Dame (9 pts)

On veut écrire une fonction qui rend une liste des cases accessibles à une dame dont on connait la position. Les fonctions qu'on vous demande d'écrire dans cet exercice n'ont pas besoin de connaitre l'état de l'échiquier.

Toutes les positions sont exprimées en coordonnées (des couples d'entiers, pas de notation algébrique). On vous demande d'écrire des fonctions intermédiaires pour faciliter l'écriture du mouvement dame.

Mouvement ligne et colonne (3 pts)

- Ecrire une fonction `mouv_lig` qui prend en paramètre une paire de coordonnées et rend la **liste** des coordonnées de toutes les cases qui ont le même numéro de ligne que celui donné dans `coord`, en excluant la position `coord`.
- De même, écrire une fonction `mouv_col` qui prend en paramètre une paire de coordonnées et rend la **liste** des coordonnées de toutes cases qui ont le même numéro de colonne que celui donné dans `coord`, en excluant la position `coord`.

Mouvement diagonal (5 pts)

- Ecrire une fonction `mouv_diag` qui prend en paramètre une paire de coordonnées et rend la **liste** des coordonnées des cases accessibles en diagonale et anti-diagonale à partir de `coord`, en excluant `coord`.

Mouvement Dame (1 pt)

- A l'aide des trois fonctions précédentes, écrire une fonction `mouv_dame` qui prend en paramètre la position de la dame et rend la liste des cases qu'elle peut atteindre, en excluant sa propre position.

Tests

```
cases = mouv_dame( (2,3) )
print(cases)
[(2, 0), (2, 1), (2, 2), (2, 4), (2, 5), (2, 6), (2, 7),
 (0, 3), (1, 3), (3, 3), (4, 3), (5, 3), (6, 3), (7, 3),
 (0, 5), (0, 1), (1, 4), (1, 2), (3, 2), (3, 4), (4, 1),
 (4, 5), (5, 0), (5, 6), (6, 7), (7, 8)]
```

4. Déterminer si des dames peuvent se prendre (7 pts)

On se pose la question suivante : *étant donné plusieurs dames placées sur l'échiquier, peuvent elles se prendre ?*

Saisie des positions des dames (3 pts)

> *Note : si vous êtes à cours de temps, passez direcement à la question suivante.*

Ecrire une fonction `saisie` qui prend en paramètre un entier `n` qui représente le nombre de dames à positionner dans l'échiquier

La fonction demande à l'utilisateur de saisir la position d'une dame en *notation algébrique*, `n` fois.

Elle retourne la liste des positions en notation matrice, c'est-à-dire après avoir converti chacune des notations algébriques en coordonnées matrice.

Détection de collision (4 pts)

Ecrire une fonction `collision`. Une collision indique qu'une dame peut en prendre une autre. La fonction prend en paramètre la liste des positions des dames en coordonnées matrice. Cette liste provient de la saisie précédente. La fonction retourne `True` si la position d'au moins une des dames est égale aux positions que peuvent atteindrent les autres dames, `False` sinon.

> *Note: si vous avez sauté la question précédente, vous pouvez définir directement les deux listes suivantes de positions :*
>
> ```
> # la liste de coordonnées correspondant à "A4", "B2", "G5", "F7"
> lst1 = [(4, 0), (6, 1), (3, 6), (1, 5)]
>
>
> # la liste de coordonnées correspondant à "A3", "G5", "B2"
> lst2 = [(5, 0), (3, 6), (6, 1)]
> ```

Test :

```
for lst in [lst1, lst2, []]:
  print('Pour', lst)
  if collision(lst):
    print("Les dames se prennent !")
  else:
    print("Les dames ne se prennent pas !")
```

Il n'y a pas de collision avec `lst1` et avec la liste vide, il y a une collision (les dames se prennent) avec `lst2`.