MPSI 1 Lycée Fénelon Sainte-Marie
Année 2014 / 2015

TP : Problème du cavalier (bis)

I Introduction

Le module a été "nettoyé" dans le sens où :
- la variable globale `N` gère la taille de la grille
- Il n'y a plus qu'une seule variable globale gérant l'évolution de la partie : `HISTORIQUE`.
- Si `HISTORIQUE` vaut `[(0,0),(1,2),(3,3),(4,2)]` par exemple, alors la position actuelle est `(4,2)` et on en est au quatrième tour. Autrement dit, la position est donnée par `HISTORIQUE[-1]` et le nombre de tours par `len(HISTORIQUE)`.
- L'affichage est gérer uniquement par `afficher()`, qui calcule tout seul quoi écrire et où.
- Un retour en arrière est matérialisé par `HISTORIQUE.pop()`, ce qui raccourcit la taille de `HISTORIQUE` de un mouvement. La nouvelle position est bien sûr `HISTORIQUE[-1]`. En revanche, on interdit les retours en arrière au-delà de la position initiale.
- L'affichage se fait en fin de tour.
- Le programme est encore dans une version assez jeune : la boucle ne s'arrête jamais.

Les fonctions sont maintenant :
- `afficher()` : affiche la grille ;
- `proposer()` : en fonction de la position et de l'historique, renvoie la liste des mouvements possibles ;
- `presenter()` : présente à chaque tour les différentes options à l'utilisateur ;
- `choisir()` : choisit un mouvement valide en fonction de la réponse de l'utilisateur ;
- `course()` : gère les différents programme au fur et à mesure des tours.

II Les mots clé `try:`, `except:`, `pass`,

Les mots-clés `try:` et `except:` sont présents en deux endroits dans le programme. Ils travaillent toujours par paire, de la façon suivante :
- Python tente d'exécuter le bloc suivant `try:`.
- Si tout va bien, le bloc suivant `except` est ignoré.
- A l'inverse, si une erreur est détectée, au lieu d'arrêter le programme et d'afficher un message d'alerte,
  - Python arrête l'exécution du bloc `try`
  - et exécute immédiatement le bloc suivant le `except`.

C'est tellement pratique que l'on cherche parfois volontairement à provoquer des messages d'erreurs. (cf. mot `assert`, prochain paragraphe).

III Code

```
from random import randint  # Module importé


N = 6; HISTORIQUE = []      # Variables globales


def afficher():
    """ Dessine la grille  n x n """

    liste = [' .'] * (N**2)
    for i in range(len(HISTORIQUE)):
        (x,y) = HISTORIQUE[i]
        liste[x + y * N] = i

    Espace = "\n" * 15
    Titre = "TOUR {}. Cavalier en {}.\n\n".format(len(HISTORIQUE), (x,y))
    L1 = " " + ("|{:2} " * N)[1:] + "\n"
    L2 = " " + ("+---"   * N)[1:] + "\n"
    Grille = (L1 + (L2 + L1) * (N-1)).format(*liste)

    print(Espace + Titre + Grille)



def proposer(x, y):
    """ Retourne sous forme de liste l'éventail des possibles. """
    poss  = []
    for dx,dy in [(1,2),(1,-2),(-1,2),(-1,-2),(2,1),(2,-1),(-2,1),(-2,-1)]:
        if 0 <= x+dx < N and 0 <= y+dy < N and (x+dx, y+dy) not in HISTORIQUE:
            poss.append((dx,dy))
    return poss


def presenter(poss):
    """ Présenter les choix à l'utilisateur. """
    prop  = "CHOIX :"                   + "\n"
    prop += "    espace : arrière"      + "\n"
    prop += "    entrée : automatique"  + "\n"
    prop += "    autre  : hasard"       + "\n"

    for i in range(8):
        if i < len(poss):
            prop += "         {}:{}\n".format(i, poss[i])
        else:
            prop += "\n"
    print(prop)
```

```
def choisir(poss):
    """ Choisir le mouvement. """
    reponse = input(" ? ")
    if reponse == "": return poss[0] # Choix automatique :
    if reponse in "_ ": return (0,0) # On demande un retour en arrière
    try:
        return poss[int(reponse)]          # Choix de l'utilisateur
    except:
        return poss[randint(0, len(poss)-1)]  # Choix au hasard



def course(x=0, y=0):
    """ Faire avancer le cavalier autant que possible. """
    HISTORIQUE.append((x,y))
    afficher()

    while 1:
        (x,y) = HISTORIQUE[-1]
        poss = proposer(x, y)

        if poss == []:
            input("BLOQUE ! Seul choix possible : arrière." + "\n" * 13)
            (dx,dy) = (0,0)                  # on est coincé, donc : retour en arrière
        else:
            presenter(poss)
            (dx,dy) = choisir(poss)

        if (dx, dy) == (0,0):               # Retour en arrière
            if len(HISTORIQUE) > 1:         # Seulement si c'est possible !
                HISTORIQUE.pop()
        else:
            HISTORIQUE.append((x+dx, y+dy))

        afficher()


if __name__ == '__main__': #==============MAIN==============
    print("Par défaut, la taille de la grille est de 6 * 6.")
    print("   - pour garder cette valeur, appuyer sur entrée,")
    p = input("   - sinon, proposer une valeur : ")
    try:
        assert 2 < int(p) < 12
        N = int(p)
    except:
        pass
    course()
```

IV Le mot-clé `assert`

Le mot clé `assert` fonctionne de la façon suivante : on le fait suivre d'un test. Si le test est validé, rien ne se passe. Si le test est `False`, une erreur se déclenche. Si l'on se trouve à l'intérieur d'un bloc `try:`, l'exécution passe directement au bloc suivant `except`. Par exemple, dans le `MAIN`, à la ligne `assert 2 < int(p) < 12`, une erreur est déclenchée si `int(p)` ne parvient à être exécuté (ici, cela se produit si `p` n'est pas un nombre écrit dans une chaîne de caractères) ou si ce nombre n'est pas compris entre 2 et 12. Si une erreur se déclenche, elle est interceptée et le bloc suivant le `except` est lu. Il contient une ligne : `pass`. Ce mot est un mot « bouche trou », car en réalité, on veut qu'il ne se passe rien ! Mais comme `try:` et `except` ne marchent qu'en binôme, et comme `except` doit impérativement être suivi d'un bloc indenté, la commande `pass` jouera ce rôle de bloc indenté.

V Questions

1. Modifier légèrement le programme `course` afin qu'à tout moment dans la boucle `while` soit définie une variable `last_move` valant toujours `(0,0)`, sauf si le dernier mouvement a été une marche arrière. Dans ce cas, `last_move` doit valoir le mouvement `(dx, dy)` que l'on vient d'annuler.

2.a) Modifier la fonction `afficher` pour qu'après avoir annulé un mouvement, deux points `" :"` apparaissent sur la case que l'on a désertée. Le calcul de cette case devra passer par la variable `last_move` transmise à `afficher`. Ces deux points disparaitront au coup d'après sans qu'il soit nécessaire de s'en occuper puisqu'en principe, `last_move` reviendra à la valeur `(0,0)`.

2.b) Modifier la fonction `présenter` pour qu'après avoir annulé un mouvement, une étoile apparaisse à gauche du mouvement qui a été annulé. Il sera sans doute utile de savoir que `liste.index(x)` renvoie le premier entier `i` tel que `liste(i)=x`. Attention, `liste.index(x)` provoque une erreur si `x` n'est pas dans `liste`.

3. Modifier la fonction `choisir` pour que si l'on appuie juste sur entrée, le mode « automatique » choisisse bêtement le choix 0 (comme c'est le cas actuellement), sauf si `last_move` vaut autre chose que `(0,0)`. Si `last_move` vaut autre chose que `(0,0)`, il faudra alors rechercher dans `poss` l'indice `i` auquel était proposé le déplacement égal à `last_move`. Il faudra ensuite que l'ordinateur choisisse le choix `i+1` si celui-ci est disponible. Sinon, il faudra revenir en arrière.

4. Modifier la fonction `choisir` et la fonction course pour que si l'utilisateur tape `"f"` ou `"F"`, le programme affiche `Fin` et se termine. Il faudra entre autre sortir de la boucle infinie, à l'aide d'un `break` par exemple.

5. Cette question demande plus de travail. Selon Wikipedia, si à chaque coup, on choisit le mouvement qui minimise le nombre de choix que l'on obtiendra au prochain coup, on trouve une façon de passer par toutes les cases (ce qui est possible dès que $N \geqslant 5$). Ajouter un mode « intelligent » choisissant les mouvements du cavalier selon ce principe.