Mastermind : Algorithmes & programme

Grandes lignes

L'ordinateur choisit une combinaison de 4 couleurs prises parmi 6 avec répétition possible
Tant que l'on n'a pas écoulé les 12 vies ou de l'obtention de 4 noirs :
    Demander la proposition du joueur
    Analyser la position du joueur (combien de pions blancs et de pions noirs à placer sur le côté)
    Afficher toutes les propositions faites jusqu'ici avec leur analyse
Afficher le résultat final : échec ou victoire + nombre de vies restantes

La partie analyse était la plus compliquée du programme
En voici le détail :

Créer une liste de commentaires constituée de 4 zéros.
Les variables blancs et noirs sont initialisées à zéro

Pour compter les noirs

Parcourir la liste des lettres de la proposition, si la lettre correspondante de la liste de solutions est la même alors le nombre noir augmente d'une unité, et la valeur correspondante de la liste de commentaire passe de 0 à 1.

Pour compter les blancs

Parcourir la liste des lettres de la proposition :
    Parcourir la liste des lettres de solution :
        En cas (d'égalité entre les lettres) et de (commentaire associé à la lettre de solution valant 0) :
            Augmenter les blancs d'une unité
            Assigner 0 au commentaire associé à la lettre solution
            Break

Remarque :
pour parcourir les trois listes : propositions, solutions et commentaires, si on utilise « for lettre in solutions : » par exemple on va bien avoir successivement toutes les lettres de la liste (ou du mot) solution mais on ne connaitra pas le rang de celles-ci, ce qui peut être gênant si l'on veut être en mesure de distinguer si elle est bien placée ou non. Il vaudra mieux faire tourner les boucles avec des indices numériques : rang dans la liste de l'élément considéré

la partie de l'algorithme d'analyse dédiée au blancs deviendra alors :

```
pour i allant de 0 à 4 exclu :
    pour j allant de 0 à 4 exclu :
        si proposition[i]==solution[j] et commentaire[j]==0 :
            augmenter blancs de 1
            commentaire[j]=1
            break
```

Pour la dernière partie des grandes lignes : l'affichage
Visiblement il faut garder en mémoire toutes les propositions et leur analyse, pour que tout soit montré sur l'écran à chaque fois qu'on arrive à la partie affichage.
On pourrait créer « historique » une liste vide au début du programme et qui à la fin de chaque analyse se verra augmentée de la liste ne contenant que le triplé : (proposition, noirs, blancs)

Pour la phase d'affichage il nous suffira de parcourir l'historique et à chaque triplet on affichera une ligne contenant la proposition, suivie d'un affichage clair du nombre de noirs et du nombre de blanc.

Cette nouvelle version permet de gérer bien des problèmes discutés en classe. Mais il ne faut pas s'endormir sur ses lauriers, il est tout à fait possible qu'il y ait des situations qui ne soient pas bien gérées. Quand on veut tester les

contre exemples, il vaut mieux mettre après le bloc créant la combinaison de manière aléatoire, une ligne remplaçant la solution par une combinaison intéressante que l'on aura choisi.

En mettant à l'épreuve l'algorithme avec la solution ABCC
J'ai eu la drôle de surprise de voir apparaitre deux types d'erreurs :

1) En proposant « J J J C » l'ordinateur me donne un noir et un blanc au lieu d'un noir et zéro blanc.
2) En proposant « CDDD » l'ordinateur me donne 0 noirs et 2 blancs alors qu'il y a zéro noir et un blanc.

Tentez d'expliquer ce qui s'est passé à l'intérieur de l'ordinateur qui a provoqué de tels affichages :

Aide pour le 2) Gestion du break :

qu'est ce que la commande « break » est censé interrompre ?
est ce qu'elle est bien placé pour cela ?

Pour gérer ce problème et vous proposer une version définitive de l'algorithme je me suis contenté d'introduire une autre ligne de contrôle pour les éléments de la proposition.
A chaque fois qu'on repérera un noir ou un blanc potentiel, on s'assurera que ni l'élément solution ni l'élément proposition concernés n'ont été utilisés auparavant.
Du coup on n'a plus besoin de la commande break.
On pourra remplacer la liste « commentaire » par les listes « com_sol » et « com_prop »

```
pour i allant de 0 à 4 exclu :
    pour j allant de 0 à 4 exclu :
        si proposition[i]==solution[j] et com_sol[j]==0 et com_prop[i]==0 :
            augmenter blancs de 1
            com_sol[j]=1
            com_prop[i]=1
```

Séance du 13/11/2020

Programmation fonctionnelle

Nous sommes finalement restés sur le chapitre 4 qui est dédié aux fonctions.
L'idée est la suivante : pour gagner en clarté et rendre votre code plus facile à tenir à jour, il est très intéressant de casser le programme en plusieurs unités aux utilités bien délimitées. Et ça, ça sera fait à l'aide de fonctions.

On utilise des fonctions sans arrêt en programmation. On peut utiliser celles qui se trouvent dans le noyau de Python ou dans des librairies associée (random, math, tkinter, pygame, turtle,…) comme : input(message), len(chaine de caractère ou liste), shuffle(liste), print(message), cos(angle), sin(angle). On peut aussi créer nous-même nos propres fonctions pour gérer les situations que l'on rencontre.
Le modèle de codage de fonction est le suivant :

def NomDeLaFonction(variables de la fonction s'il y en a) :
Instructions
…
Instructions
return résultatsObtenus *(ligne facultative)*

En classe on a fait quelques exemples de fonctions utiles :

```
def tirage():
    #cette fonction sans variable va nous donner un mot de quatre
    lettres prises au hasard parmi les lettres possibles (chaque lettre
    correspond à une des couleurs que l'on peut jouer)
    potentiel=["J","A","B","V","N","R"]#liste des lettres potentielles
    mot=""  #au départ le mot est vide
    for i in range(4):  #à quatre reprises
        shuffle(potentiel)  #on mélange les lettres de la liste
        mot+=potentiel[0] #on ajoute au mot une nouvelle lettre (la
        première de la liste)
    return mot  #on donne le mot obtenu (il faudra que lorsqu'on
    appelle la fonction tirage(), on soit prêt pour exploiter le mot,
    par exemple en le stockant dans une variable)
```

```
def actionJ():
    # cette fonction va demander encore et encore une proposition au
    joueur jusqu'à ce que celui ci réponde un mot de quatre lettres
    valide=False  #valide indique si on a un mot contient le bon nombre
    de lettre, vu que l'utilisateur n'a fait encore aucune proposition,
    la réponse est NON !
    while not valide : # not valide est le contraire de valide, donc si
    cette variable vaut False alors "not valide" vaut True et vice
    versa. ici on veut que la boucle tourne tant que valide soit faux,
    on aurait pu écrire aussi "while valide==False :"
        proposition=input("votre proposition") #on pose une question et
        on stocke la réponse
        if len(proposition)==4 : #si la longueur vaut quatre
            valide = True #on indique que la réponse est enfin valide
    #quand la réponse est enfin valide la boucle arrête de tourner
    return proposition  #et la fonction donne au programme la
    proposition acceptable
```

Les élèves devaient créer une fonction comptant le nombre de jetons noirs (autrement dit le nombre de cas où un jeton de couleur est de la bonne couleur au bon endroit)

```
def jetonsNoirs(solution, proposition):
    #cette fonction a deux paramètres, j'ai choisi des noms explicites
    pour aider les gens qui vont lire mon code à l'utiliser au mieux.
    noir=0 # au début je n'ai aucun jeton noir
    for i in range(4) : # quatre fois de suite je vais faire :
        if solution[i]==proposition[i]: #je regarde si la lettre de rang
        i de la solution et celle de même rang de la proposition sont
        identiques
            noir+=1 #si c'est le cas on a un jeton noir de plus
    return noir
```

Cette fonction sera hélas inutile car on a besoin de déterminer le nombre de jetons noirs et de jetons blanc coup sur coup à l'aide de liste de contrôle comme on a pu le voir dans l'algorithme.

Pour dimanche les élèves du groupe 1 doivent m'envoyer la fonction Analyse(solution, proposition) qui me renverra deux valeurs : le nombre de jetons noir et le nombre de jetons, j'ai proposé une version de l'aide que je reformule ici en utilisant les notations de l'algorithme proposé un peu plus tôt dans le document

```
def Analyse(solution, proposition):
    #on recycle la fonction jetonsNoirs en incluant les listes
    permettant de savoir si une lettre a déjà été utilisée
    noir=0 # au début je n'ai aucun jeton noir
    blanc=0 # et aucun jeton blanc
    com_sol=[0,0,0,0] #c'est la liste commentant l'état des lettres
    constituant la solutions
    com_prop=[0,0,0,0] #c'est la liste commentant l'état des lettres
    constituant la solutions

    for i in range(4) : # quatre fois de suite je vais faire :
        if solution[i]==proposition[i]: #je regarde si la lettre de rang
        i de la solution et celle de même rang de la proposition sont
        identiques
            noir+=1 #si c'est le cas on a un jeton noir de plus
            com_sol[i]=1 #on indique que la i ème lettre de la solution est
            maintenant utilisée et donc on ne pourra la prendre en compte
            quand on cherchera les blancs
            com_prop[i]=1 #idem pour i ème lettre de la proposition
    # à votre tour de jouer et de compter les jetons blancs
    # en utilisant l'algorithme du document




    return noir,blanc
```

Les exemples précédents sont observables puis copiables à l'adresse :
https://repl.it/@jkergot/fonctions-pour-mastermind#main.py

solution pour la fonction Analyse(solution, proposition)

```
def Analyse(solution, proposition):
  #on recycle la fonction jetonsNoirs en incluant les listes permettant de
  savoir si une lettre a déjà été utilisée
  noir=0 # au début je n'ai aucun jeton noir
  blanc=0 # et aucun jeton blanc
  com_sol=[0,0,0,0] #c'est la liste commentant l'état des lettres
  constituant la solutions
  com_prop=[0,0,0,0] #c'est la liste commentant l'état des lettres
  constituant la solutions
  for i in range(4) : # quatre fois de suite je vais faire :
    if solution[i]==proposition[i]: #je regarde si la lettre de rang i de
    la solution et celle de même rang de la proposition sont identiques
      noir+=1 #si c'est le cas on a un jeton noir de plus
      com_sol[i]=1 #on indique que la i ème lettre de la solution est
      maintenant utilisée et donc on ne pourra la prendre en compte quand
      on cherchera les blancs
      com_prop[i]=1 #idem pour i ème lettre de la proposition
  # à votre tour de jouer et de compter les jetons blancs
  # en utilisant l'algorithme du document
  for i in range(4) :
    for j in range(4) :
      if proposition[i]==solution[j] and com_sol[j]==0 and com_prop[i]==0 :
        #pour que l'on ait un blanc il faut que les lettres coincident et
        qu'elles n'aient pas été utilisées
        blanc+=1
        com_sol[j]=1  #important pour éviter d'utiliser une nouvelle fois
        cette lettre
        com_prop[i]=1 #idem
  return noir,blanc #la fonction Analyse renvoi 2 information pour les
  récupérer il faudra écrire quelque chose de la forme black,white =
  Analyse(solutionG, propositionJ )
```

vendredi 20 les élèves du groupe 2 ont travaillé sur une fonction affichage

```
def affichage(memory) :
    #affiche le contenu de la liste memory d'une manière adaptée
    # si la variable valait memoire=[("JBNV",0,2),("JBVN",0,2),("NVJB",
    1,1),("NBJV",2,0)] alors la fonction afficherait :
    #+------------------+
    #|01|JBNV|  0  |  2  |
    #+------------------+
    #|02|JBVN|  0  |  2  |
    #+------------------+
    #|03|NVJB|  1  |  1  |
    #+------------------+
    #|04|NBJV|  2  |  0  |
    #+------------------+
```

Cet exercice permettait de recycler le travail du premier devoir maison avec le drapeau créé avec une boucle et des structures conditionnelles.
Les difficultés étaient les suivantes :

- envisager le tableau sous forme de boucle
- Se rappeler comment est ce que l'on fait pour accéder à des éléments dans un tableau, puis dans un tableau contenant un autre tableau (ou comme c'est le cas ici un triplet / tuple)
- Arriver à concaténer proprement les différents éléments : il valait mieux le faire en dehors du print car ce dernier à tendance à rajouter des espaces inattendus.
- Difficultés bonus : pour concaténer les éléments on se retrouvait à ajouter des chaines de caractères et des nombres (le nombre de noirs, le nombre de blancs) et là ça bloque. L'opération de concaténation ne marche qu'avec des éléments de même natures (des chaines de caractères entre elles, des listes entre elles)
    - Solution : pour convertir un nombre en chaine de caractère on utilise la fonction str(nombre)

Remarque :
En passant dans les rangs pour aider, ça pouvait être rapide et efficace avec certains mais d'autres n'avaient pas du tout intégré les bases vues et détaillées en classe du coup pour les aider il aurait fallu passer trois fois plus de temps avec eux ce qui n'est pas juste pour les personnes qui attendent. c'est surtout très gênant pour la suite : sans bases toutes les nouvelles connaissances vont s'effriter rapidement et donc ne tiendront jamais dans le temps. Les exercices sur franceioi devraient leur permettre de rattraper leur retard pour peu qu'ils se bougent

Correction de la fonction affichage

```
def affichage(memory) :
  #affiche le contenu de la liste memory d'une manière adaptée
  # si la variable valait memoire=[("JBNV",0,2),("JBVN",0,2),("NVJB",1,1),
  ("NBJV",2,0)] alors la fonction afficherait :
  #+------------------+
  #|01|JBNV|  0  |  2  |
  #+------------------+
  #|02|JBVN|  0  |  2  |
  #+------------------+
  #|03|NVJB|  1  |  1  |
  #+------------------+
  #|04|NBJV|  2  |  0  |
  #+------------------+
  print("+------------------+")
  for i in range(len(memory)) :
    if i<9 : # si le numéro de l'essai n'a pas de dizaine ça veut dire que
    le chiffre des dizaines vaut zéro, il faut imposer l'affichage de ce
    chiffre.
      ligne="|0"+str(i+1)
    else :
      ligne="|"+str(i+1)
    ligne+="|"+memory[i][0]+"|  "+str(memory[i][1])+"  |  "+str(memory[i]
    [2])+"  |"
    print(ligne)
    print("+------------------+")
  print("         noirs blancs")
  print("")
```

Pour finir votre programme mastermind

Vous pouvez faire les choses à votre sauce avec vos notations du moment que votre programme aboutit par contre si vous vous sentez bloqués vous pouvez écrire la courte série d'instructions suivantes à la suite de toutes les fonctions qui ont été vues jusqu'ici

solutionG = tirage()
black et essai seront fixés à 0
played la liste des coups joués et de leur analyse est pour l'instant une liste vide
tant que le nombre de noir est plus petit que 4 et que le nombre d'essai est plus petit ou égal à 12
  propositionJ=actionJ()
  black, white = Analyse(solutionG, propositionJ)
  played est augmentée de la liste contenant seulement le triplet (propositionJ ,black, white)
  on affiche les coups joués et leur analyse à l'aide de la fonction affichage(memory) utilisée avec la variable "played"
  on augmente le nombre d'essai effectué de 1
une fois que la boucle a cessé de tourner si black vaut 4 on écrit victoire sinon on écrit échec
on écrira aussi le nombre de coups joués

vous trouverez un repl à compléter qui vous permettra de finir Mastermind :
https://repl.it/@jkergot/DerniereLigneDroite

.