Jeu du SNAKE

Le menu est bien, sauf qu'il ne sert pas à grand-chose : aucun réglage n'est proposé (on pourrait éventuellement changer la couleur du fond, du serpent, de la pomme ou bien entrer son nom ou que sais-je), on peut juste cliquer sur un bouton pour commencer le jeu ou quitter sans avoir joué (!).
La fenêtre de jeu qui s'ouvre a une taille ridicule comparée à celle du menu : pourquoi ne pas avoir dimensionné cette fenêtre à l'aide des paramètres d'écran lus pour afficher le menu ? D'une façon générale, la répartition des tâches entre les deux associés n'est pas égale, ce qui est acceptable, mais la coordination n'est pas excellente : chacun s'est occupé de sa tâche et les deux ne sont pas vraiment corrélées ce qui est regrettable. Le menu, s'il existe doit avoir une autre fonction que de proposer de jouer.

Le dynamisme du jeu est très bon, très simple. Les paramètres sont bien réglés (taille initiale du serpent, vitesse, apparition des pommes, affichage du score) ce qui permet une réelle expérience de jeu.

Les commentaires sont trop rares et pas assez explicites : au début du programme je tombe sur i=0 et je me demande ce que fait cette variable peu explicite. Ce n'est qu'en voyant qu'elle est utilisée partout que je comprends qu'elle sert à indiquer la direction. Il aurait fallu en écrire l'utilité (i=3 vers la droite, i=2 vers le bas ou i=1 vers le haut…) et éventuellement lui donner un nom plus parlant comme « dir » par exemple.
Cette remarque est valable pour les autres variables : sc par exemple est initialisée à 0… on pourrait penser que c'est un entier… et bien non, un peu plus loin sc est affectée à une étiquette (label) de score. À quoi servent les variables f, g, h… ces noms ne sont pas du tout parlants, ils servent au développeur qui les prend dans l'ordre où il en a besoin, mais qui va en oublier la fonction quelques jours plus tard. Il faut toujours penser que c'est dans l'intérêt de tous d'avoir des variables aux noms clairs et des commentaires additionnels pour clarifier encore leur usage.

L'utilisation de l'attribut global est massif pour toutes les fonctions même celles où il n'est pas nécessaire. Global permet de modifier une variable externe à la fonction, si on ne la modifie pas dans la fonction, global n'est pas nécessaire car une variable globale est toujours accessible. Pour la fonction jeu() par exemple, il est écrit global score,sc,cdrH,cdrW,x,y (6 variables globales) alors qu'il suffit d'écrire global sc puisque seule sc est modifiée par cette fonction.
D'une façon générale, il est conseillé d'éviter au maximum de manipuler des variables globales (cela peut engendrer des conflits).
Des aspects importants des fonctions n'ont pas été, du tout, utilisés : le passage d'arguments aux fonctions et le renvoi par une fonction d'une valeur de retour. Ces aspects permettent de manipuler des variables extérieures à une fonction avec une plus grande sécurité.

L'utilisation des listes est curieuse, du fait d'une sous-utilisation des méthodes de listes.
Pour faire avancer le serpent vers le haut, par exemple, on utilise la fonction :

```
def up(event): #définissent la direction du serpent quand on appuie sur une flèche
    global x,y,i,h
    i=1
    direction.reverse()
    direction.append(i)
    del direction[0]
    direction.reverse()
```

L'inversion des éléments avec reverse() sert à supprimer le dernier élément (la queue qui passe momentanément au rang 0), ajouter un élément à la tête (momentanément dernier élément) et une nouvelle application de reverse() remet tout dans l'ordre (tête au rang 0). On obtient le même résultat en faisant :

```
def up(event): #redéfini la direction du serpent et place la tête dans la bonne direction
    global i
    i=1
    direction.insert(0,i)
    del direction[-1]
```

La fonction corps pourrait être simplifiée, comme d'autres fonctions, puisqu'on y fait presque la même chose dans les 4 directions. (Le commentaire qui accompagne cette fonction n'est pas clair : que fait-elle vraiment ? Elle allonge le corps du serpent ou elle le fait avancer?)

```
def corps(): #fonction qui definie la creation d'un carré du sepent en fonction de la direction de ce dernier
    global i,body,x,y,dot_x,dot_y,g,ser
    if direction[-1]==1:
            ser=intface.create_rectangle(dot_x[g],dot_y[g]+10,dot_x[g]+10,dot_y[g]+20,fill='black',outline='#34C924')
            x_dot=dot_x[g]
            y_dot=dot_y[g]+10
            dot_x.append(x_dot)
            dot_y.append(y_dot)
            body.append(ser)
            direction.append(1)
```

… même chose dans les 3 autres directions + la direction 0 (!) à quoi sert cette direction qui est définie comme a direction 4 ?

Il suffit de remplacer, sauf erreur de ma part, toute cette fonction (43 lignes) par ces 14 lignes :

```
def corps(): #fonction qui ajoute un élément au serpent
  global g
  x_dot=dot_x[g]
  y_dot=dot_y[g]
  if direction[-1]==1: y_dot+=10
  elif direction[-1]==2: y_dot-=10
  elif direction[-1]==3: x_dot+=10
  else: x_dot-=10
  dot_x.append(x_dot)
  dot_y.append(y_dot)
  ser=intface.create_rectangle(x_dot,y_dot,x_dot+10,y_dot+10,fill='black',outline='#34C924')
  body.append(ser)
  direction.append(direction[-1])
  g=g+1
```

La fonction anim() gagnerait aussi beaucoup à être simplifiée selon le même principe (qu'est-ce qui est réellement différent dans les différents mouvements du serpent ?)

Par contre, l'utilisation des listes est astucieuse pour faire avancer le serpent puisqu'en ajoutant juste un élément à la tête et en supprimant l'élément de queue, on obtient exactement ce que l'on souhaite, sans toucher au reste du serpent.

Projet de note : 18/20

Détail de la note :

- préprojet : 2/2
- compte-rendu : 3.5/4 (narration intéressante des difficultés rencontrées pour le dynamisme du serpent, les solutions adoptées)
- Programme : 12.5/14 (très fonctionnel, simple, assez bonne utilisation des fonctions et des listes, bonne prise en main de tkinter, coordonner davantage le menu au jeu)