Partie 1

Programmer un Snake sur Python-Pygame

Ce projet consiste à realiser (en partie) le jeu *Snake*. Il traite des types abstraits personnalisés. Le *Snake* est un genre de jeu vidéo dans lequel le joueur dirige un serpent qui grandit et constitue ainsi lui-même un obstacle.

Le script ci-dessous donne les éléments de base du programme.

Repérer (directement sur le script) les différentes instructions propres au module Pygame :

- import de la librairie
- initialisation de l'objet `pygame`
- initialisation de la fenêtre graphique
- lecture des evênements (interaction avec le clavier)
- dessiner un rectangle (carré) sur l'écran
- mettre à jour l'affichage
- attendre 30ms

Repérer ensuite :

- la déclaration de variables globales
- la boucle principale et sa condition d'execution
- l'instruction qui génère un déplacement
  - horizontal (modification d'une coordonnée)
  - vertical

1.1 Creation d'un type abstrait `serpent`

Le *serpent* est un objet qui *stocke les données* d'une certaine manière, et qui propose quelques fonctions utiles pour interragir avec lui (*une interface*). Il s'agit donc d'un *type abstrait*.

Pour représenter le serpent, comme dans le jeu *snake*, les coordonnées des pixels du serpent seront stockées dans une Liste `[(x1,y1),(x2,y2),...]`.

1.1.1 Interface

Vous allez programmer chaque fonction de l'interface à partir des renseignements suivants. Ces fonctions seront placées dans le proggramme, avant la boucle principale.

- fonction `affiche`

```
def affiche(S):
    # affiche tous les pixels du serpent
    # a partir de leurs coordonnees dans S
    # utilise une boucle for
```

L'affichage de chaque pixel du serpent necessite une boucle bornée :

```
for coord in serpent:
    x = coord[0]
    y = coord[1]
    pygame.draw.rect(dis, black, [x, y, 10, 10])
```

- fonction `tete`

```
def tete(S):
   # retourne les coordonnees x, y de la tete du serpent
   # qui sont stockees dans le dernier element S[-1]
   # sous la forme d'un tuple (x,y)
```

- fonction `ajoute_tete` Il peut être necessaire d'ajouter un pixel au serpent. Celui-ci s'ajoutera à la fin de la liste de coordonnées :

```
def ajoute_tete(S, x, y):
    # ajoute un tupple (x,y) a la fin de la liste S
```

- fonction `supprime_queue`

A chaque déplacement, cela ajoute un pixel à la fin de la liste. Il s'agit de la tête du serpent. Il faut alors retirer le premier pixel (`serpent[0]`) si l'on veut que sa longueur reste constante.

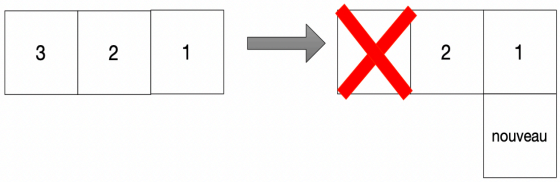

FIGURE 1 – deplacement du serpent vers le bas

```
def supprime_queue(S):
    # decale toutes les valeurs de la liste S vers la gauche:
    # a l'aide d'une boucle bornee sur les indices i:
    # copie toutes les valeurs S[i+1] dans S[i]
    # puis supprime le dernier element
```

1.1.2 Dans le programme `main.py`

Modifier le script pour interagir avec le type abstrait *serpent*.

Lire les consignes sur le TP en ligne

1.2 Créer un module

Mettre les fonctions de l'interface `tete`, `ajoute_tete`, et `supprime_queue` dans un nouveau fichier, `utils.py`.

Lire les consignes sur le TP en ligne

Partie 2

Questions

1. Expliquer : Comment le programme gère les déplacements du serpent ?
2. Cette gestion des déplacements, est-elle facilitée par l'interface du type *serpent* ? Si oui, comment ?
3. Quelles fonctionnalités du jeu manque-t-il pour que celui-ci soit complet ?
4. Si vous deviez créer le jeu complet :

- quelles fonctions ajouteriez vous à l'interface serpent ?
- Quelle autre structure de données envisageriez-vous ?

Partie 3

Annexe - script principal

```
import pygame

pygame.init()

white = (255, 255, 255)
black = (0, 0, 0)
red = (255, 0, 0)

clock = pygame.time.Clock()
dis = pygame.display.set_mode((800, 600))

game_over = False

x1 = 300
y1 = 300

x1_change = 0
y1_change = 0

while not game_over:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                x1_change = -10
                y1_change = 0
            elif event.key == pygame.K_RIGHT:
                x1_change = 10
                y1_change = 0
            elif event.key == pygame.K_UP:
                y1_change = -10
                x1_change = 0
            elif event.key == pygame.K_DOWN:
                y1_change = 10
                x1_change = 0

    x1 += x1_change
    y1 += y1_change
    dis.fill(white)
    pygame.draw.rect(dis, black, [x1, y1, 10, 10])

    pygame.display.update()
    clock.tick(30)


pygame.quit()
quit()
```