

Programmer un Snake sur Python-Pygame

Ce projet consiste à réaliser (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 événements (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'exécution
- l'instruction qui génère un déplacement
 - horizontal (modification d'une coordonnée)
 - vertical

1.1 Création 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 interagir 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 $[(x1, y1), (x2, y2), \dots]$.

1.1.1 Interface

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

- fonction affiche

```

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

```

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

```

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

```

- fonction tete

```

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

```

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

```

1 def ajoute_tete(S, x, y):
2     # ajoute un tuple (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.

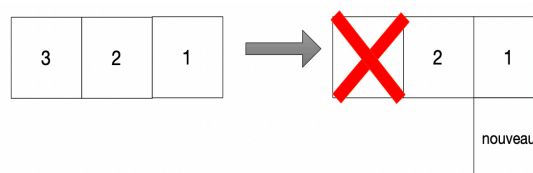


FIGURE 1 – déplacement du serpent vers le bas

```

1 def supprime_queue(S):
2     # decale toutes les valeurs de la liste S vers la gauche:
3     # a l'aide d'une boucle bornee sur les indices i:
4     # copie toutes les valeurs S[i+1] dans S[i]
5     # 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 ?

Annexe - script principal

```
1 import pygame
2
3 pygame.init()
4
5 white = (255, 255, 255)
6 black = (0, 0, 0)
7 red = (255, 0, 0)
8
9 dis = pygame.display.set_mode((800, 600))
10
11 game_over = False
12
13 x1 = 300
14 y1 = 300
15
16 x1_change = 0
17 y1_change = 0
18
19 while not game_over:
20     for event in pygame.event.get():
21         if event.type == pygame.QUIT:
22             game_over = True
23         if event.type == pygame.KEYDOWN:
24             if event.key == pygame.K_LEFT:
25                 x1_change = -10
26                 y1_change = 0
27             elif event.key == pygame.K_RIGHT:
28                 x1_change = 10
29                 y1_change = 0
30             elif event.key == pygame.K_UP:
31                 y1_change = -10
32                 x1_change = 0
33             elif event.key == pygame.K_DOWN:
34                 y1_change = 10
35                 x1_change = 0
36
37         x1 += x1_change
38         y1 += y1_change
39         dis.fill(white)
40         pygame.draw.rect(dis, black, [x1, y1, 10, 10])
41
42         pygame.display.update()
43         clock.tick(30)
44
45
46 pygame.quit()
```