Projet Snake
Correction

Christophe Viroulaud

Première - NSI

Eval 10

Dans le déroulement d'un jeu, on ne sait généralement pas quand l'utilisateur va appuyer sur une touche. Un programme uniquement **séquentiel** n'est pas adapté. Il faut alors pouvoir réagir à un événement : on parle de **programmation événementielle**.

Comment mettre en place une programmation événementielle ?

Sommaire

Conception générale

- Initialiser le plateau.
- Initialiser le serpent.
- Démarrer la boucle de jeu :
  - Effacer l'écran.
  - Créer les pommes (si besoin).
  - Dessiner les pommes.
  - Avancer le serpent.
  - Manger la pomme (éventuellement).
  - Dessiner le serpent.
  - Si le serpent ne sort pas et s'il ne se croise pas :
    - recommencer la boucle de jeu.
  - Sinon :
    - finir le jeu.

```
1  def jeu(fenetre, canevas, serpent: list, pommes: list):
2      canevas.delete("all")
3      creer_pomme(pommes)
4      dessiner_pommes(pommes)
5      avancer_serpent(serpent)
6      manger_pomme(serpent, pommes)
7      dessiner_anneaux(serpent)
8      if not verif_croisement(serpent) and \
9          not verif_sortie(serpent):
10         fenetre.after(500, jeu, fenetre, canevas, serpent,
       pommes)
11     else: # fin du jeu
12         canevas.create_text(DIM*TAILLE//2, DIM*TAILLE//2,
13                             text="GAME OVER",
14                             font="Arial 36",
15                             tag="gameover")
```

Code 1 – Implémentation

Sommaire

Programme principal - Initialisations

```
1  # imports
2  import tkinter
3  from tkinter import ttk
4  from random import randint
5
6  TAILLE = 20
7  DIM = 30
8  NB_POMMES = 10
9  # N E S O
10 DEPLACEMENTS = ((0, -1), (1, 0), (0, 1), (-1, 0))
11
12 # fonctions
```

Code 2 – Initialisations et imports

```
1  # principal
2  fenetre = tkinter.Tk()
3  fenetre.title("Snake")
4
5  canevas = tkinter.Canvas(fenetre,
6                           width=TAILLE*DIM,
7                           height=TAILLE*DIM)
8  canevas.pack()
9
10 # dernière ligne du programme: met à jour les
       variables
11 fenetre.mainloop()
```

Code 3 – La biblitothèque **tkinter** crée une interface graphique pour Python.

Sommaire

Lancement du jeu


```
1 pommes = []
2 initial = TAILLE//2
3 serpent = [{"col": initial, "lig": initial, "val": 0},
4              {"col": initial+1, "lig": initial, "val": 1}]
5 dpct = DEPLACEMENTS[1]
6
7 jeu(fenetre, canevas, serpent, pommes)
```

Code 4 – On initialise le serpent dans le programme principal.

Observation

Le serpent est un tableau de dictionnaires.

Sommaire

Écouteurs


```
1 fenetre.bind("<Right>", deplacer_snake)
2 fenetre.bind("<Left>", deplacer_snake)
3 fenetre.bind("<Up>", deplacer_snake)
4 fenetre.bind("<Down>", deplacer_snake)
```

Code 5 – Placer un écouteur.

À retenir

La méthode `bind` place un écouteur sur certaines touches.
La fonction `deplacer_snake` ne sera exécutée que quand la touche sera pressée.
On parle de programmation **événementielle**.

```
1  def deplacer_snake(event):
2      """
3      effectue le changement de direction
4      pas de retour arrière possible
5      """
6      global dpct
7      if event.keysym == "Up" and dpct !=
       DEPLACEMENTS[2]:
8          dpct = DEPLACEMENTS[0]
9      elif event.keysym == "Right" and dpct !=
       DEPLACEMENTS[3]:
10         dpct = DEPLACEMENTS[1]
11     elif event.keysym == "Down" and dpct !=
       DEPLACEMENTS[0]:
12         dpct = DEPLACEMENTS[2]
13     elif event.keysym == "Left" and dpct !=
       DEPLACEMENTS[1]:
14         dpct = DEPLACEMENTS[3]
```

Code 6 – La fonction sera appelée à chaque événement.

Sommaire

Conception détaillée - Découpage

Chaque action énoncée dans la conception générale sera réalisée par une fonction.

> **Activité 1 :** Télécharger et extraire l'annexe **`snake-annexe.zip`**

Sommaire

Conception détaillée - Créer les pommes

Activité 2 : La fonction `creer_pomme` crée un nouveau nombre à **une place libre du plateau**. Compléter la fonction.

Correction

```
1  def creer_pomme(pommes: list) -> None:
2      if len(pommes) < NB_POMMES:
3          # création pomme
4          pomme = (randint(0, TAILLE-1),
5                   randint(0, TAILLE-1))
6          # vérification superposition de 2 pommes
7          while pomme in pommes:
8              pomme = (randint(0, TAILLE-1),
9                       randint(0, TAILLE-1))
10         # ajout pomme
11         pommes.append({"col": pomme[0],
12                        "lig": pomme[1],
13                        "val": randint(0, 99)})
```

Sommaire

Avancer le serpent

Activité 3 : Pour avancer le serpent chaque anneau (dictionnaire) récupère les coordonnées de l'anneau précédent. Enfin la tête est déplacée en fonction des valeurs de `dpct`.
Compléter la fonction.

Correction

```
1  def avancer_serpent(serpent: list) -> None:
2      for i in range(len(serpent)-1, 0, -1):
3          serpent[i]["col"] = serpent[i-1]["col"]
4          serpent[i]["lig"] = serpent[i-1]["lig"]
5
6      # nouvelles coordonnées de la tête (avec
       changement éventuel)
7      serpent[0]["col"] = serpent[0]["col"]+dpct[0]
8      serpent[0]["lig"] = serpent[0]["lig"]+dpct[1]
```

Sommaire

Insérer une pomme

Activité 4 : Il est inutile d'effectuer un tri complet du serpent quand il mange un nombre : il suffit d'insérer le nouveau nombre jusqu'à sa bonne position. Le choix est fait d'ajouter le nombre **à la fin** du serpent puis de le faire remonter vers la tête.

1. Compléter la fonction `inserer` qui décale le dernier nombre jusqu'à sa position, en remontant le tableau.
2. Compléter alors la fonction `manger_pomme`.

Correction


```
1  def inserer_pomme(serpent: list) -> None:
2      i = len(serpent)-1
3      while i > 0 and serpent[i]["val"] < serpent[i-1]["val"]:
4          # inversion des 2 anneaux consécutifs
5          serpent[i]["val"], serpent[i-1]["val"] = \
6              serpent[i - 1]["val"], serpent[i]["val"]
7          i = i-1
```

```
1  def manger_pomme(serpent: list, pommes: list) -> None:
2      # position tête du serpent
3      col = serpent[0]["col"]
4      lig = serpent[0]["lig"]
5
6      for i in range(len(pommes)):
7          # vérifie si la tête est sur une pomme
8          if pommes[i]["col"] == col and \
9              pommes[i]["lig"] == lig:
10             # supprime la pomme mangée du tableau
11             pomme = pommes.pop(i)
12             # récupère le sens de la queue
13             dx = serpent[len(serpent)-1]["col"] -
14                     serpent[len(serpent)-2]["col"]
15             dy = serpent[len(serpent)-1]["lig"] -
16                     serpent[len(serpent)-2]["lig"]
```

Code 7 – Première partie de la fonction

```
1            # ajoute la pomme à la fin du serpent
2            serpent.append(
3                {"col": serpent[len(serpent)-1]["col"]+dx,
4                 "lig": serpent[len(serpent)-1]["lig"]+dy,
5                 "val": pomme["val"]})
6
7            # positionne la pomme dans le serpent
8            inserer_pomme(serpent)
9
10           # la pomme est placée, inutile de parcourir la
   fin du tableau
11           break
```

Code 8 – Fin de la fonction

Sommaire

Fin de partie

```
1  def verif_croisement(serpent: list) -> bool:
2      for i in range(1, len(serpent)):
3          if serpent[i]["col"] == serpent[0]["col"]
       and \
4                  serpent[i]["lig"] == serpent[0]["
     lig"]:
5              return True
6      return False
```

Code 9 – Il reste à vérifier si le serpent ne se croise pas.

```
1 def verif_sortie(serpent: list) -> bool:
2     if serpent[0]["col"] < 0 or \
3             serpent[0]["col"] >= TAILLE or \
4             serpent[0]["lig"] < 0 or \
5             serpent[0]["lig"] >= TAILLE:
6         return True
7     return False
```

Code 10 – Et qu'il ne sorte pas des limites.