Projet

Projet

Pygame : Initiation

1. Preambule

Pygame est un package de Python facilitant la création de jeux basés une interface graphique. Vous pouvez :

- l'installer sur votre distribution Python, par `pip3 install pygame`.
- le tester directement via https://repl.it/, en choisissant `pygame` dans la liste des langages proposés.

🐍 **Script Python**

```
import pygame, sys
from pygame.locals import *

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

while continuer:
    for event in pygame.event.get():
```

```
        if event.type == KEYDOWN:
          if event.key == K_RIGHT:
            continuer = False

    pygame.display.flip()

pygame.quit()
```

Commentaires

- Durant tout le code, notre scène de travail sera l'objet `ecran`, dans lequel nous viendrons coller de nouveaux éléments.

Éléments structurants d'un code `pygame` :

- `pygame.init()` effectue une initialisation globale de tous les modules `pygame` importés. À mettre au début du code.
- `while continue :` comme très souvent dans les jeux, la structure essentielle est une boucle infinie dont on ne sortira que par un appui sur la flèche bas où continue passe en `False`.

2. Apparition d'un personnage

2.1. Téléchargement de l'image

Nous allons travailler avec le sprite ci-dessous, nommé `perso.png`.

Téléchargez-le pour le mettre dans le même dossier que votre code `pygame`. A redéfinir avec une bonne dimension.

Vous pouvez trouver sur internet un grand nombre de sprites libres de droits, au format `png` (donc gérant la transparence), dans de multiples positions (ce qui permet de simuler des mouvements fluides). Ici nous travaillerons avec un sprite unique.

2.2. Importation de l'image dans la fenêtre

🐍 **Script Python**

```
perso = pygame.image.load("Paragoomba.png").convert_alpha()
```

La fonction `convert_alpha()` est appelée pour que soit correctement traité le canal de transparence (canal *alpha*) de notre image.

2.3. Affichage de l'image

À ce stade, `perso` est un objet `pygame` de type `Surface` .

Afin de facilement pouvoir le déplacer, nous allons stocker la position de cet objet dans une variable `position_perso`, qui sera de type `rect`.

🐍 Script Python

```
position_perso = perso.get_rect()
```

Pour afficher cette image, nous allons venir le superposer aux éléments graphiques déjà dessinés (en l'occurence : rien) avec l'instruction `blit()` :

🐍 Script Python

```
fenetre.blit(perso, position_perso)
```

▸ récapitulatif du code

🐍 Script Python

```
import pygame, sys
from pygame.locals import *

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

perso = pygame.image.load("Paragoomba1.png").convert_alpha()
position_perso = perso.get_rect()



while continuer:
    ecran.blit(perso, position_perso)
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_RIGHT:
                continuer = False

    pygame.display.flip()

pygame.quit()
```

3. Gestion des évènements

En informatique, un événement peut être une entrée clavier (soit l'appui soit le relâchement d'une touche), le déplacement de votre souris, un clic (encore une fois, appui ou relâchement, qui seront traités comme deux événements distincts). Un bouton de votre joystick peut aussi engendrer un événement, et même la fermeture de votre fenêtre est considéré comme un événement !

Pour Pygame, un événement est représenté par un type et divers autres attributs que nous allons détailler dans ce chapitre.

De plus, il faut savoir que chaque événement créé est envoyé sur une file (ou queue), en attendant d'être traité. Quand un événement entre dans cette queue, il est placé à la fin de celle-ci. Vous l'aurez donc compris, le premier événement transmis à Pygame sera traité en premier ! Cette notion est très importante, puisque nous allons nous en servir sous peu !

Ce type de queue est dit FIFO.

3.1. Comment les capturer ?

On utilise le module event de Pygame.

Voici ce que nous dit la documentation à propos de ce module :

Afficher/Masquer la documentation

- pygame.event
- pygame module for interacting with events and queues
- pygame.event.pump — internally process pygame event handlers
- pygame.event.get — get events from the queue
- pygame.event.poll — get a single event from the queue
- pygame.event.wait — wait for a single event from the queue
- pygame.event.peek — test if event types are waiting on the queue
- pygame.event.clear — remove all events from the queue
- pygame.event.event_name — get the string name from and event id
- pygame.event.set_blocked — control which events are allowed on the queue
- pygame.event.set_allowed — control which events are allowed on the queue
- pygame.event.get_blocked — test if a type of event is blocked from the queue
- pygame.event.set_grab — control the sharing of input devices with other applications
- pygame.event.get_grab — test if the program is sharing input devices
- pygame.event.post — place a new event on the queue
- pygame.event.Event — create a new event object
- pygame.event.EventType — pygame object for representing SDL events
- event

Et comme on peut le voir, le module event ne permet pas que d'intercepter des événements. Il nous permet aussi de créer des événements. Et même d'en bloquer !

- Lorsqu'un programme `pygame` est lancé, la variable interne `pygame.event.get()` reçoit en continu les évènements des périphériques gérés par le système d'exploitation.
- Nous allons nous intéresser aux évènements de type `KEYDOWN` (touche de clavier appuyée) ou de type `MOUSEBUTTONDOWN` (boutons de souris appuyé).

3.2. Évènements clavier

3.2.1. Exemple de code

La structure de code pour détecter l'appui sur une touche de clavier est, dans le cas de la détection de la touche «Flèche droite» :

🐍 **Script Python**

```
for event in pygame.event.get():
  if event.type == KEYDOWN:
    if event.key == K_RIGHT:
      print("flèche droite appuyée")
```

La touche (en anglais *key*) «Flèche Droite» est appelée `K_RIGHT` par `pygame`.

Le nom de toutes les touches peut être retrouvé à l'adresse pygame ref

Remarque : c'est grâce à la ligne initiale

🐍 **Script Python**

```
from pygame.locals import *
```

que la variable `K_RIGHT` (et toutes les autres) est reconnue.

3.2.2. Problème de la rémanence

Quand une touche de clavier est appuyée, elle le reste un certain temps. Parfois volontairement (sur un intervalle long) quand l'utilisateur décide de la laisser appuyée, mais aussi involontairement (sur un intervalle très court), lors d'un appui «classique».
Il existe donc toujours un intervalle de temps pendant lequel la touche reste appuyée. Que doit faire notre programme pendant ce temps ? Deux options sont possibles :

- **option 1 :** considérer que la touche appuyée correspond à un seul et unique évènement, quelle que soit la durée de l'appui sur la touche.
- **option 2 :** considérer qu'au bout d'un certain délai, la touche encore appuyée doit déclencher un nouvel évènement.

Par défaut, `pygame` est réglé sur l'option 1. Néanmoins, il est classique pour les jeux vidéos de vouloir que «laisser la touche appuyée» continue à faire avancer le personnage. Nous allons donc faire en sorte que toutes les 50 millisecondes, un nouvel appui soit détecté si la touche est restée enfoncée. Cela se fera par l'expression :

🐍 **Script Python**

```
pygame.key.set_repeat(50)
```

3.3. Évènements souris

3.3.1. Exemple de code

La structure de code pour détecter l'appui sur un bouton de la souris est, dans le cas de la détection du bouton de gauche (le bouton 1) :

🐍 **Script Python**

```
for event in pygame.event.get():
  if event.type == MOUSEBUTTONDOWN and event.button == 1 :
     print("clic gauche détecté")
```

3.3.2. Récupération des coordonnées de la souris

Le tuple `(abscisse, ordonnée)` des coordonnées de la souris sera récupéré avec l'instruction `pygame.mouse.get_pos()` .

3.4. Déplacement du personnage

Le déplacement d'un personnage se fera toujours par modification de ses coordonnées (et visuellement, par effacement de la dernière position).
Ce déplacement pourra être :
- absolu : on donne de nouvelles coordonnées au personnage.
- relatif : on indique de combien le personnage doit se décaler par rapport à sa position initiale.

3.5. Déplacement absolu

Pour afficher le personnage à la position `(100,200)`, on écrira :

🐍 **Script Python**

```
position_perso.topleft = (100,200)
```

où `position_perso` est l'objet de type `rect` contenant les coordonnées.

Exo

Coder un script pour déplacer Paragoomba à la souris (Paragoomba doit toujours suivre la souris) (`MOUSEMOTION`)

Correction

🐍 **Script Python**

```
import pygame, sys
from pygame.locals import *
from random import randint

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

perso = pygame.image.load("Paragoomba1.png").convert_alpha()
position_perso = perso.get_rect()
position_perso.topleft = (100,200)

while continuer:
  pygame.draw.rect(ecran, (10,186,181), (0, 0, 640, 480))
  for event in pygame.event.get():
    if event.type == pygame.MOUSEMOTION:
      position_perso = event.pos
      if event.type == KEYDOWN:
        if event.key == K_RIGHT:
          continuer = False
  ecran.blit(perso, position_perso)
  pygame.display.flip()

pygame.quit()
```

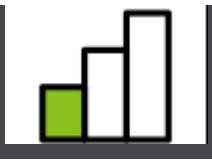
Exo

Coder un script pour dessiner un rectangle sur l’écran au relâchement d’un bouton de la souris.

Correction

🐍 **Script Python**

```
import pygame, sys
from pygame.locals import *
from random import randint

pygame.init()

#ecran = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
ecran = pygame.display.set_mode((640, 480))
ecran.fill([10,186,181])

continuer = True

perso = pygame.image.load("Paragoomba1.png").convert_alpha()
position_perso = perso.get_rect()
position_perso.topleft = (100,200)

largeur = 10
hauteur = 10
couleur = (200, 80, 20)

while continuer:
 for event in pygame.event.get():
  if event.type == pygame.MOUSEBUTTONUP:
   x, y = event.pos
   pygame.draw.rect(ecran, couleur, (x, y, largeur, hauteur))
  if event.type == KEYDOWN:
   if event.key == K_RIGHT:
    continuer = False
 ecran.blit(perso, position_perso)
 pygame.display.flip()
```



4. Le jeu de tennis (à la Pong)

Pour la création du jeu de tennis, nous allons organiser notre code en plusieurs fichiers :

- un fichier tennis.py qui sera le programme principal
- un fichier constantes.py qui contiendra les constantes utilisées par les autres fichiers (hauteur et largeur de fenêtre, couleurs), certaines fonctions...

4.1. Les packages utilisés

On commence par introduire les packages (bibliothèques) qui seront utilisées :

🐍 **Script Python**

```
import pygame
from pygame.locals import *
from constantes import *
```

4.2. Les constantes du jeu : fichier constantes.py

On définit la hauteur et la largeur de la fenêtre ainsi que l'abscisse du mur qui sera situé à droite.

On définit également un jeu de couleurs.

Fichier constantes.py

🐍 **Script Python**

```
import pygame
from pygame.locals import *
from constantes import *


# initialisation de l'écran de jeu
pygame.init()



police = pygame.font.SysFont("Arial", 25)
fonte = pygame.font.Font(None, 30)


# Initialise la fenêtre de jeu
largeur_ecran = 600
hauteur_ecran = 400

screen = pygame.display.set_mode((largeur_ecran,hauteur_ecran))
pygame.display.set_caption("Tennis")

# variables d'état

hauteur_raquette=50

largeur_raquette =10
dist_mur   = 20  # distance du mur au bord de la raquette


raquette_G_x = dist_mur
raquette_G_y = 50


ball_x = int(largeur_ecran / 2)
ball_y = int(hauteur_ecran / 2)
ball_speed_x = -4
ball_speed_y = -4
ball_rayon  = 10

score = 0
vie=2

COORD_X_MUR = largeur_ecran-20


# Definit des couleurs RGB
BLACK = [0, 0, 0]
WHITE = [255, 255, 255]
GREEN = [24, 161, 80]
```

```
RED   = [255, 0, 0]
BLUE  = [30 , 36 , 161]
ORANGE = [196 , 92 , 54]

#fonctions permettant de dessiner la balle et les deux raquettes
def Raquette(x, y):
  R = (x,y,largeur_raquette,hauteur_raquette)
  pygame.draw.rect(screen, WHITE, R, 0)

def Balle(x,y):
  pygame.draw.circle(screen, WHITE, (x,y),10, 0)

def Mur():
  R = (COORD_X_MUR,0,20,hauteur_ecran)
  pygame.draw.rect(screen, GREEN, R, 0)



def touche_clavier():
  for event in pygame.event.get():
    if event.type == KEYDOWN:
      # Ctrl-C pour quitter le jeu
      if event.key == pygame.K_c and pygame.key.get_mods() & pygame.KMOD_CTRL:
        quitter()
      # retourner la touche pressée
      return event.key
    # sinon, ne rien retourner (valeur nulle)
    return None

def attente():
  while touche_clavier() == None:
      pygame.display.update()

# initialisation de l'écran de jeu
pygame.init()




def affiche_texte_centre(texte, y=-1, couleur=None):
  if couleur == None:
    couleur = ORANGE
  rendu = fonte.render(texte, True, couleur)
  rectangle = rendu.get_rect()
  if y == -1:
    rectangle.center = ((largeur_ecran) / 2 , (hauteur_ecran) / 2)
  else:
    rectangle.center = ((largeur_ecran) / 2 , y)
  screen.blit(rendu, rectangle)

def affiche_texte(texte, x, y, couleur=None):
  if couleur == None:
    couleur = WHITE
  rendu = fonte.render(texte, True, couleur)
  rectangle = rendu.get_rect()
```

```
    rectangle.center = (x, y)
    screen.blit(rendu, rectangle)
```

4.3. Le jeu

On crée la fenêtre de jeu, on utilise la fonte courante et on charge les sons qui seront utilisé pour le jeu :

- la musique d'ambiance music.mp3
- et le bruit de verre brisé glass_break.wav qui indique la fin du jeu

Le jeu se compose de trois parties :

- l'écran d'accueil qui indique quelles sont les touches pour jouer
- le jeu de tennis
- la fin de partie qui affiche le score obtenu par le joueur

tennis.py

🐍 **Script Python**

```python
import pygame
from pygame.locals import *
from constantes import *









# Gestion du rafraichissement de l'écran
clock = pygame.time.Clock()

# Cache le pointeur de la souris
pygame.mouse.set_visible(0)




############################################################
# écran d'accueil
############################################################
screen.fill(BLACK)
affiche_texte_centre("Appuyez sur une touche pour commencer", 100)
affiche_texte_centre("Flèche haut pour faire monter la raquette", 140)
affiche_texte_centre("Flèche bas pour faire descendre la raquette", 170)

attente()

# Le jeu continue tant que l'utilisateur ne ferme pas la fenêtre
Termine = False

# Boucle principale de jeu
while not Termine:
  # recupère la liste des évènements du joueur
  event = pygame.event.Event(pygame.USEREVENT)

  # dessine le mur de droite

  # EVENEMENTS
  # détecte le clic sur le bouton close de la fenêtrepygame.Rect
  for event in pygame.event.get():
      if event.type == pygame.QUIT:
        Termine = True

  # récupère la liste des touches claviers appuyeées sous la forme d'une liste de booléens
  KeysPressed = pygame.key.get_pressed()
```

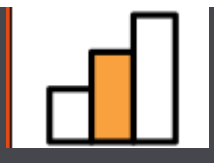
```
# LOGIQUE
# déplacement du palet gauche

if KeysPressed[pygame.K_UP]:
    raquette_G_y -= 3

if KeysPressed[pygame.K_DOWN]:
    raquette_G_y += 3


if raquette_G_y < 0 :
    raquette_G_y = 0


if raquette_G_y > hauteur_ecran - hauteur_raquette :
    raquette_G_y = hauteur_ecran - hauteur_raquette

# Déplacement de la balle
ball_x += ball_speed_x
ball_y += ball_speed_y

if ball_y < ball_rayon or ball_y > hauteur_ecran - ball_rayon :
    ball_speed_y *= -1

# collision entre la balle et le palet de gauche
if ball_x  <  dist_mur + largeur_raquette + ball_rayon :
    if ball_y > raquette_G_y  and  ball_y  <  raquette_G_y + hauteur_raquette :
        ball_speed_x *= -1
        score+=1



# collision avec les murs gauche et droit
if ball_x < ball_rayon :
    ball_x = int(largeur_ecran / 2)
    ball_y = int(hauteur_ecran / 2)
    vie-=1


if ball_x > largeur_ecran - ball_rayon :
    ball_speed_x *= -1



# AFFICHAGE
# Dessine le fond
screen.fill(BLACK)
Mur()
Raquette(raquette_G_x, raquette_G_y)

Balle(ball_x,ball_y)

#  dessine le texte dans une zone de rendu à part
texte = "Votre score : " + str(score) + " Vie : " + str(vie)
if score == 15 :
```

```
            texte = "Joueur GAGNANT"
            Termine=True
        if vie<=0:
            texte = 'PERDU'
            Termine=True


        zone = police.render( texte, True, GREEN)
        # affiche la zone de rendu au dessus de fenetre de jeu
        screen.blit(zone,(280,10))



        # Bascule l'image dessinée à l'écran
        pygame.display.flip()

          # Demande à pygame de se caler sur 30 FPS
        clock.tick(30)

    screen.fill( 'black')
    texte = "Votre score est de {} points".format(score)
    affiche_texte_centre(texte,150)
    affiche_texte_centre("Appuyez sur une touche pour terminer", 200)
    attente()

    # Ferme la fenêtre
    del(police)
    pygame.quit()
```

4.4. Le jeu Pong en lui-même

A vous

Faites votre propre jeu avec une deuxième raquette, meilleur gestion des rebonds, changement de vitesses....

5. SNAKE en Python, le plus simplement possible 

5.1. Version 0

5.1.1. Pygame

On importe pygame avec :

🐍 **Script Python**

```
import pygame
from pygame.locals import *
```

Le second import sert à quitter le jeu propremement.

5.1.2. Constantes

On crée quelques constantes :

🐍 **Script Python**

```
HAUTEUR = 600  # hauteur de la fenetre
LARGEUR = 600  # largeur de la fenetre
BLOC = 20

# Les couleurs utilisées
NOIR = (..., ..., ...)  # fond
ROUGE = (..., ..., ...) # pomme
JAUNE = (..., ..., ...)  # tête
VERT = (..., ..., ...)  # corps
CYAN = (..., ..., ...)  # texte

FPS = 30
```

5.1.3. initialisation

On initialise le jeu :

🐍 **Script Python**

```
pygame.init()
horloge = pygame.time.Clock()
fenetre = pygame.display.set_mode((LARGEUR, HAUTEUR))
pygame.display.set_caption('Snake')

pygame.display.update()
```

5.1.4. Boucle Infinie

Tous les jeux comportent une boucle infinie. Celle-ci ne contient pas grand chose :

- quitter le jeu,
- remplir la fenêtre de noir
- faire avancer l'horloge
- mettre à jour les affichages

5.1.5. Boucle infinie

🐍 **Script Python**

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
```

```
            pygame.quit()
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                pygame.quit()
    fenetre.fill(NOIR)
    horloge.tick(FPS)
    pygame.display.update()
```

5.1.6. Boucle infinie

- La boucle `for event in...` permet de récupérer les événements "cliquer sur la croix" ou "appuyer sur Escape" et quitte le jeu dans ce cas.
- Ensuite on dessine la fenêtre, remplie de noir
- On fait avancer l'horloge
- On affiche tout ça

5.2. Version 1

5.2.1. Ecrire du texte

Cette fonction nous permettra d'écrire facilement le score

🐍 Script Python

```
def drawText(text, font, surface, x, y):
    textobj = font.render(text, 1, CYAN)
    textrect = textobj.get_rect()
    textrect.topleft = (x, y)
    surface.blit(textobj, textrect)
```

5.2.2. Taille de la police, valeur du score

🐍 Script Python

```
font = pygame.font.SysFont(None, 48)
```

et

🐍 Script Python

```
score = 0
```

5.2.3. Le serpent

Le serpent est une double liste

🐍 Script Python

```
snake = [[3, 3], [2, 3], [1, 3]]
```

Le premier élément est sa tête, elle est en `[3,3]` ensuite vient son corps. Il commence donc avec une taille de 3.

5.2.4. Dessiner le serpent

Dans la boucle infinie, avant l'horloge :

🐍 Script Python

```
for elt in snake[1:]:
    pygame.draw.rect(fenetre, VERT, (elt[0] * BLOC, elt[1] * BLOC,  BLOC, BLOC))
pygame.draw.rect(fenetre, JAUNE, (snake[0][0] * BLOC, snake[0][1] * BLOC, BLOC, BLOC))
```

Le corps est vert et la tête jaune.

5.2.5. Afficher le score

On utilise notre fonction crée plus tôt :

🐍 Script Python

```
drawText(str(score), font, fenetre, 0.2*LARGEUR, 0.2*HAUTEUR)
```

5.3. Version 2

On ne capturait que "Escape" et le clic sur la croix. On ajoute les flêches.

5.3.1. Capturer les touches du jeu

5.3.2. Diminuer la vitesse de rafaîchissement

🐍 Script Python

```
FPS = 5
```

5.3.3. Déplacer le serpent

On commence par créer une direction (= la vitesse) en dehors de la boucle infinie

🐍 Script Python

```
direction = (1, 0)
```

5.3.4. Déplacer le serpent

Chaque pression d'une flêche change la direction :

🐍 Script Python

```
    key = pygame.key.get_pressed()
    if key:
        if key[pygame.K_UP]:
```

```
        direction = (..., ...)
    if key[pygame.K_DOWN]:
        direction = (..., ...)
    if key[pygame.K_LEFT]:
        direction = (..., ...)
    if key[pygame.K_RIGHT]:
        direction = (..., ...)
```

5.3.5. Déplacer le serpent

Ensuite la tête.

C'est l'ancienne tête, qui s'est déplacée :

🐍 **Script Python**

```
head = [snake[0][0] + direction[0], snake[0][1] + direction[1]]
```

5.3.6. Déplacer le serpent

Le corps se déplace.

1. On ajoute la tête au début :

Correction

🐍 **Script Python**

```
snake.insert(0, head)
```

1. On perd un élément de fin :

Correction

🐍 **Script Python**

```
snake.pop(-1)
```

5.4. Version 4

5.4.1. La mort du serpent

Il meurt :

- s'il quitte l'écran
- si sa tête est dans son corps

5.4.2. La mort du serpent

Correction

🐍 **Script Python**

```
 if head in snake[1:] or head[0] < 0 or head[0] > LARGEUR / BLOC - 1 or head[1] < 0 or head[1] >
HAUTEUR / BLOC - 1:
  pygame.quit()
```

5.5. Version 5

5.5.1. Fluidité

Le jeu n'est pas fluide.

On va mettre à jour les éléments du jeux toutes les 1.5 secondes et afficher 30 frames par secondes.

Il nous faut deux variables supplémentaires :

1. Une valeur pour décider quand mettre à jour
2. Un compteur

5.5.2. Fluidité

🐍 **Script Python**

```
FPS = 30
MAJ = 10

# ...

# juste avant la boucle infinie
compteur = 0
```

5.5.3. Fluidité

Dans la boucle infinie

🐍 **Script Python**

```
 if compteur == MAJ:
  compteur = 0
  head = [snake[0][0] + direction[0], snake[0][1] + direction[1]]
  # mettre les autres événements concernant le snake

 # On augmente le compteur
 # tout à la fin de la boucle infinie
 compteur += 1
```

5.5.4. Nourriture

On crée d'abord une nouvelle liste :

🐍 **Script Python**

```
pomme = [8, 3]
```

5.5.5. Nourriture

On dessine la pomme comme la tête, mais en rouge

Correction

🐍 **Script Python**

```
pygame.draw.rect(fenetre, ROUGE,(pomme[0] * BLOC, pomme[1] * BLOC, BLOC, BLOC))
```

5.5.6. Nourriture

Puis on détecte la collision avec la pomme.

En cas de collision :

1. Le score augmente
2. Une nouvelle pomme est crée.

La boucle `while` empêche la pomme d'apparaître sur le serpent

5.5.7. Nourriture

Correction

🐍 **Script Python**

```
from random import randint
  # ...

  # Dans la boucle infinie
  if snake[0] == pomme:
    score += 1
    while pomme in snake:
       pomme = [randint(0, LARGEUR / BLOC - 1),
              randint(0, HAUTEUR / BLOC - 1)]
```

5.5.8. Nourriture

S'il n'y a pas de collision le serpent diminue, sinon il conserve sa taille

🐍 **Script Python**

```
else:
  snake.pop(-1)
```

5.6. Conclusion

C'est terminé...

Snake en 100 lignes (peu commentées) avec le minimum d'instructions. On peut faire beaucoup plus court mais c'est déjà très simple

- Python permet notamment de créer des jeux,
- Créer un jeu avec Pygame n'est pas difficile,
- Il nous faut quelques constantes, quelques éléments de jeu (serpent, tête)
- Une boucle infinie dans laquelle
- On lit les saisies de l'utilisateur
- On effectue les calculs (nouvelle tête, collisions etc.)
- On met à jour les éléments graphiques

Code final

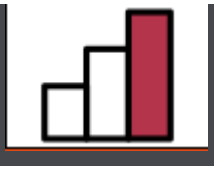
🐍 Script Python

```
"""
Snake simple
Snake réalisé "simplement" en Pygame avec Python 3.
Nécessite Pygame et Python 3.7
"""
# -*- coding: utf-8 -*-

# pour choisir où faire apparaître la nouvelle pomme
from random import randint
# la librairie pygame
import pygame
# afin de quitter le jeu proprement
from pygame.locals import *


# Les dimensions de la fenêtre
HAUTEUR = 600  # hauteur de la fenetre
LARGEUR = 600  # largeur de la fenetre
# Ainsi que celle d'un carré à l'écran : 20 pixels
BLOC = 20

# Les couleurs utilisées
NOIR = (0, 0, 0)  # fond
ROUGE = (193, 68, 51)  # pomme
JAUNE = (208, 210, 62)  # tête
VERT = (97, 195, 73)  # corps
CYAN = (51, 133, 193)  # texte

# Vitesse de rafaîchissement du jeu
FPS = 60
# On effectue les calculs toutes les 15 frames
MAJ = 15


############################################################
#####################  Fonctions  ##########################
############################################################


def drawText(text, font, surface, x, y):
    '''
    Dessine du texte à l'écran
    @param text: (str) le texte
    @param font: (pygame.font) la police
    @param surface: (pygame.surface) la surface sur laquelle écrire
    @param x, y: (int) les coordonnées du texte
    '''
    textobj = font.render(text, 1, CYAN)
    textrect = textobj.get_rect()
```

```
    textrect.topleft = (x, y)
    surface.blit(textobj, textrect)


################################################################
####################   GAME INITIALISATION   ###############
################################################################


# pygame
# les éléments indispensables sont init, time.Clock()  et un
# display
pygame.init()
horloge = pygame.time.Clock()
fenetre = pygame.display.set_mode((LARGEUR, HAUTEUR))
# titre de la fenêtre
pygame.display.set_caption('Snake')

# taille et type de la fonte
font = pygame.font.SysFont(None, 48)

# on met immédiatement à jour avant de commencer
pygame.display.update()

################################################################
####################   GAME LOOP    ############################
################################################################

# les éléments du jeu
# le serpent est un tableau à 2 dimensions.
# le premier est la tête, les suivants le corps
# chaque élément est une liste de cooordonnées [abs, ord]
snake = [[3, 3], [2, 3], [1, 3]]
direction = (1, 0)
pomme = [8, 3]

# le score est un entier
score = 0

# compteur de frame pour les mises à jour
compteur = 0

while True:
    # Saisies de l'utilisateur

    # quitter le jeu
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()

        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                pygame.quit()

    # déplacer le serpent
```

```
    key = pygame.key.get_pressed()
    if key:
        if key[pygame.K_UP]:
            # on change la direction vers le haut
            direction = (0, -1)
        if key[pygame.K_DOWN]:
            # on change la direction vers le bas
            direction = (0, 1)
        if key[pygame.K_LEFT]:
            # on change la direction vers la gauche
            direction = (-1, 0)
        if key[pygame.K_RIGHT]:
            # on change la direction vers la droite
            direction = (1, 0)

    # Calculs
    # ils ne sont effectués que toutes les 15 frames
    if compteur == MAJ:
        # on reset le compteur
        compteur = 0

        # la nouvelle tête est l'ancienne, déplacée dans la direction
        head = [snake[0][0] + direction[0], snake[0][1] + direction[1]]

        # on l'insère au début
        snake.insert(0, head)

        # collision tête / pomme
        if snake[0] == pomme:
            # on augmente le score
            score += 1
            # on déplace la pomme en dehors du corps
            while pomme in snake:
                # nécessaire de tirer plusieurs fois si on n'a
                # pas de chance !
                pomme = [randint(0, LARGEUR / BLOC - 1),
                         randint(0, HAUTEUR / BLOC - 1)]
        else:
            # si le serpent n'a pas mangé la pomme, il diminue
            snake.pop(-1)

        # mort du serpent
        # s'il touche son corps
        # ou s'il quitte l'écran
        if head in snake[1:] \
                or head[0] < 0 \
                or head[0] > LARGEUR / BLOC - 1 \
                or head[1] < 0 \
                or head[1] > HAUTEUR / BLOC - 1:
            pygame.quit()

    # Graphiques

    # d'abord on remplit de noir pour cacher les images précédentes
    fenetre.fill(NOIR)
```

```
    # Ensuite on dessine le corps en vert
    for elt in snake[1:]:
        pygame.draw.rect(fenetre, VERT,
                         (elt[0] * BLOC, elt[1] * BLOC, BLOC, BLOC))

    # la tête en jaune
    pygame.draw.rect(fenetre, JAUNE,
                     (snake[0][0] * BLOC, snake[0][1] * BLOC, BLOC, BLOC))

    # la pomme en rouge
    pygame.draw.rect(fenetre, ROUGE,
                     (pomme[0] * BLOC, pomme[1] * BLOC, BLOC, BLOC))

    # le score dans le coin de l'écran
    drawText(str(score), font, fenetre, 0.2 * LARGEUR, 0.2 * HAUTEUR)

    # On met pygame à jour
    # en avançant l'horloge
    horloge.tick(FPS)
    # en dessinant les éléments
    pygame.display.update()
    # et comptant les frames
    compteur += 1
```

6. Mini-Tetris 

Un peu d’histoire

Tetris est un jeu vidéo entre arcade et puzzle, conçu par Alekseï Pajitnov en juin 1984. Le succès devient planétaire et tous les consoles qui suivront posséderont leur version de Tetris. Il fait partie des jeux les plus addictifs de l’époque, avec Pacman.

Code mini-tetris - Le début

🐍 **Script Python**

```
import pygame
import copy
import random

# initialisation de l'écran de jeu
pygame.init()

# Definit des couleurs RGB
NOIR  = (0, 0, 0)
VERT  = (0, 255, 0)
ROUGE = (255, 0, 0)
BLEU  = (0 , 0 , 255)
GRIS  = (128,128,128)
CYAN  = (0,255,255)
JAUNE = (255,255,0)
ORANGE= (255,150,0)
VERT  = (0,255,255)
MAUVE = (180,80,255)
LCoul = [ NOIR, GRIS, CYAN, JAUNE, MAUVE, ORANGE, BLEU, ROUGE, VERT ]

# PIECES
P_I = [ [0,2,0],
        [0,2,0],
        [0,2,0] ]

P_T = [ [0,0,0],
        [4,4,4],
        [0,4,0]]

P_O = [ [3,3,0],
        [3,3,0],
        [0,0,0] ]

P_L = [ [0,0,0],
        [5,5,5],
        [5,0,0] ]

P_J = [ [0,0,0],
        [6,6,6],
        [0,0,6]]

P_Z = [ [7,7,0],
        [0,7,7],
        [0,0,0] ]

P_S = [ [0,8,8],
        [8,8,0],
        [0,0,0]]
```

```
LP  = [ P_I, P_T, P_O, P_L, P_J, P_Z, P_S]


def AffPiece():
    P = LP[idpiece]
    for dx in range(3):
        for dy in range(3):
            c = P[dy][dx]
            if c != 0:
              idcoul = int(c)
              xx = (px+dx) * LCASE
              yy = (py+dy) * LCASE
              R = (xx,yy,LCASE,LCASE)
              pygame.draw.rect(screen,LCoul[idcoul],R)



# DECORS
LIGNE_VIDE = [1,1] + [0]*11 + [1]*2
DECOR = []
for i in range(16):
    DECOR.append(LIGNE_VIDE.copy())
DECOR.append([1]*15)
DECOR.append([1]*15)

LCASE = 20
def AfficheDecors():
    for y in range(len(DECOR)) :
        for x in range(len(DECOR[0])):
            xx = x * LCASE
            yy = y * LCASE
            id = DECOR[y][x]

            pygame.draw.rect(screen,LCoul[id],(xx,yy,LCASE,LCASE))
            pygame.draw.rect(screen,NOIR,(xx,yy,LCASE,LCASE),1)


# Initialise la fenêtre de jeu
screenWidth = 300
screenHeight = 360
screen = pygame.display.set_mode((screenWidth,screenHeight))
pygame.display.set_caption("MINI TETRIS")


# Gestion du rafraichissement de l'écran
clock = pygame.time.Clock()
# Cache le pointeur de la souris
pygame.mouse.set_visible(0)


# variables d'état
# piece courante
idpiece = 1
px  = 6
py  = 0
```

```
rot = 0
#Touches
KEyDown  = 0
KeyUp    = 0
KeyLeft  = 0
KeyRight = 0

#compteur d'affichage
comptage = 0


# Le jeu continue tant que l'utilisateur ne ferme pas la fenêtre
Termine = False

# Boucle principale de jeu
while not Termine:
  # recupère la liste des évènements du joueur
  event = pygame.event.Event(pygame.USEREVENT)

  # EVENEMENTS
  # détecte le clic sur le bouton close de la fenêtre
  for event in pygame.event.get():
     if event.type == pygame.QUIT:
       Termine = True

  # récupère la liste des touches claviers appuyeées sous la forme d'une liste de booléens
  KeysPressed = pygame.key.get_pressed()

  # LOGIQUE
  # déplacement de la pièce
  comptage += 1
  if comptage % 20 == 0 :
        py += 1


  if KeysPressed[pygame.K_UP] and KeyUp == 0:
       pass

  if KeysPressed[pygame.K_LEFT] and KeyLeft == 0:
       pass

  if KeysPressed[pygame.K_RIGHT] and KeyRight == 0:
       pass

  if KeysPressed[pygame.K_DOWN] and KeyDown == 0:
       pass


  KeyDown  = KeysPressed[pygame.K_DOWN]
  KeyUp    = KeysPressed[pygame.K_UP]
  KeyLeft  = KeysPressed[pygame.K_LEFT]
  KeyRight = KeysPressed[pygame.K_RIGHT]
```

```
    # AFFICHAGE
    # Dessine le fond
    AfficheDecors()
    AffPiece()

    # Bascule l'image dessinée à l'écran
    pygame.display.flip()

      # Demande à pygame de se caler sur 30 FPS
    clock.tick(30)

  # Ferme la fenêtre
  pygame.quit()
```

Le jeu est fonctionnel que dans une petite partie : une pièce descend mais il est impossible de la déplacer.

6.1. Présentation du code

Les constantes couleurs :

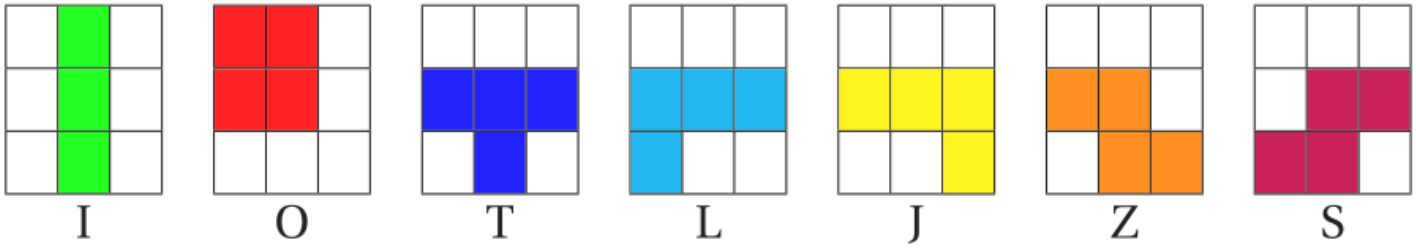
🐍 **Script Python**

```
# Definit des couleurs RGB
NOIR  = (0, 0, 0)
VERT  = (0, 255, 0)
ROUGE = (255, 0, 0)
BLEU  = (0 , 0 , 255)
GRIS  = (128,128,128)
CYAN  = (0,255,255)
JAUNE = (255,255,0)
ORANGE= (255,150,0)
MAUVE = (180,80,255)
LCoul = [ NOIR, GRIS, CYAN, JAUNE, MAUVE, ORANGE, BLEU, ROUGE, VERT ]
```

Le fond noir est associé à la valeur d'indice 0, les murs gris à la valeur d'indice 1 et chaque pièce du jeu est associée avec la couleur d'indice compris entre 2 et 8.

6.1.1. Les différentes pièces :

Pour simplifier les algorithmes, on va utiliser des combinaisons de carrés qui s'inscrivent dans une grille 3 \(\times\) 3



Toutes les pièces sont stockées dans des listes de 3 \(\times\) 3 éléments. Une valeur nulle correspond à une case vide et une valeur non nulle indique une case pleine ainsi que sa couleur. L'ensemble des pièces est stocké dans une liste nommée LP :

🐍 **Script Python**

```
# PIECES
P_I = [ [0,2,0],
        [0,2,0],
        [0,2,0] ]

P_O = [ [3,3,0],
        [3,3,0],
        [0,0,0] ]

P_T = [ [0,0,0],
        [4,4,4],
        [0,4,0]]

P_L = [ [0,0,0],
        [5,5,5],
        [5,0,0] ]

P_J = [ [0,0,0],
        [6,6,6],
        [0,0,6]]

P_Z = [ [7,7,0],
        [0,7,7],
        [0,0,0] ]

P_S = [ [0,8,8],
        [8,8,0],
        [0,0,0]]

LP  = [ P_I, P_O, P_T, P_L, P_J, P_Z, P_S]
```

Les variables d'état sont présentées ci-dessous.

🐍 **Script Python**

```
# variables d'état
# piece courante
idpiece = 1
px  = 6
py  = 0
rot = 0
#Touches
KeyDown  = 0
KeyUp    = 0
KeyLeft  = 0
KeyRight = 0
```

La variable `idpiece` indique l'indice de la pièce courante dans la liste `LP`. Ainsi le jeu démarre avec la pièce T.
Les variable `px`, `py` et `rot` indique la position en \(x\), et \(y\) da la pièce dans la grille, ainsi que sa

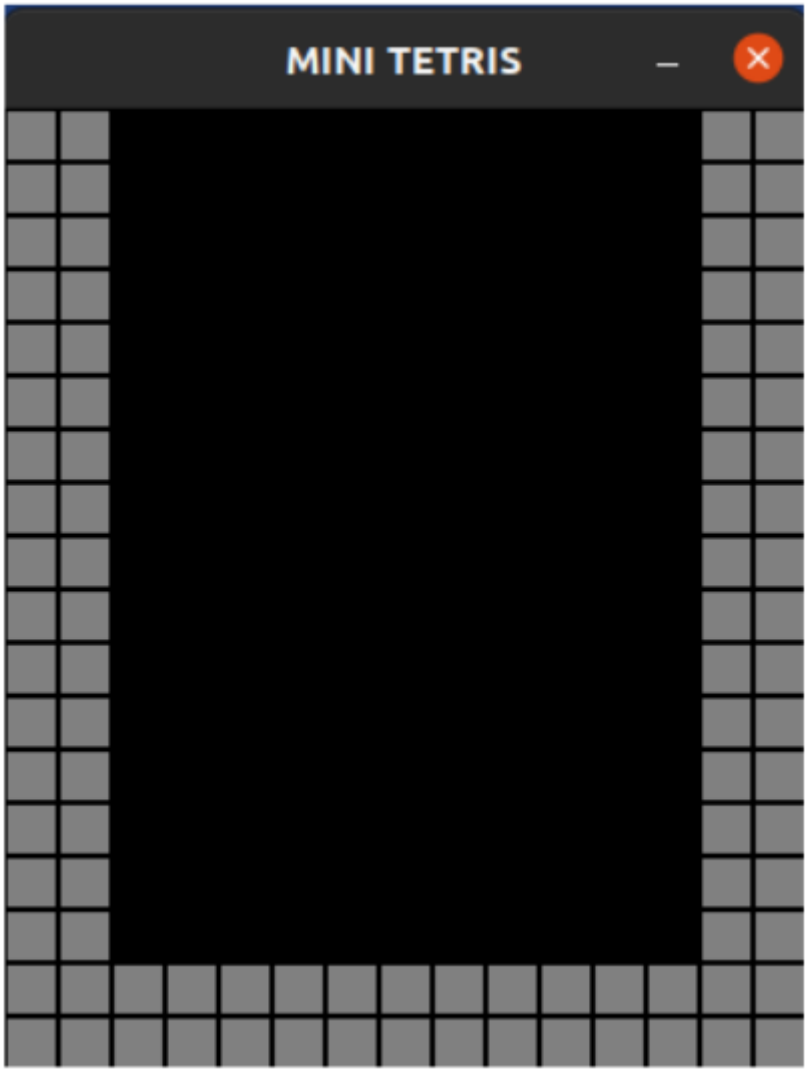
rotation : 0 pour \(0^{\circ}\) et 1 pour \(90^{\circ}\) .
Et quatre variables pour stocker l'état précédent des touches fléchées : enfoncé ou non. L'intérêt des de pouvoir détecter les appuis sur ces touches.

6.2. Affichage du décors :

🐍 **Script Python**

```
# DECORS
LIGNE_VIDE = [1,1] + [0]*11 + [1]*2
DECOR = []
for i in range(16):
    DECOR.append(LIGNE_VIDE.copy())
DECOR.append([1]*15)
DECOR.append([1]*15)


LCASE = 20
def AfficheDecors():
    for y in range(len(DECOR)) :
        for x in range(len(DECOR[0])):
            xx = x * LCASE
            yy = y * LCASE
            id = DECOR[y][x]

            pygame.draw.rect(screen,LCoul[id],(xx,yy,LCASE,LCASE))
            pygame.draw.rect(screen,NOIR,(xx,yy,LCASE,LCASE),1)
```

Le décor est stocké dans une grille de 15 cases de large pour 18 de haut. Comme la largeur des cases fait 20 pixels, on a donc une fenêtre de taille 300 \(\times\) 360 pixels. On définit une constante `LIGNE_VIDE` composée de 2 colonnes sur la gauche et sur la droite, qui marquent les bords avec des cases grises,donc de code couleur associé 1. Les 11 cases centrales vides sont remplies avec la valeur 0? Le décor est défini comme une liste de 18 lignes.
Les 16 premières sont des lignes vides, et les 2 dernières sont remplies de 1. Pour créer les 16 lignes vides, nous utilisons la liste `LIGNE_VIDE` qu'on copie avec la fonction `copy()`. Ceci est très important car chaque ligne doit être indépendante !\



Les valeurs sont stockées dans une liste de listes intitulée `DECOR`. Ainsi `len(DECOR)` correspond au nombre de lignes et `len(DECOR[0])` au nombre de colonnes du jeu.
En écrivant `DECOR[y][x]` on accède à l'indice de couleur pour la case de coordonnées (x,y). L'origine du décor (0,0) est positionnée en haut à gauche de l'écran.
La variable `LCASE` définit la largeur d'une case en pixels. Pour dessiner entièrement la grille, on utilise un double boucle en x et y.

On dessine un carré plein grâce à la première fonction `draw.rect()`, puis les bords noirs avec le deuxième appel.

6.3. Déplacement des pièces :

On déplace la pièce courante avec une technique particulière. On utilise une variable comptage qui comptabilise le nombre d'affichages effectués. Le test effectué est : `comptage % 20 == 0`, ce qui produit 20 affichages. Comme on est à 30 FPS, cela se produit toutes les 0,66 seconde. a ce moment-là, on fait descendre la pièce d'une ligne vers le bas. (A ce niveau aucune collision n'est pas gérée)

🐍 **Script Python**

```
# LOGIQUE
# déplacement de la pièce
comptage += 1
if comptage % 20 == 0 :
        py += 1
```

Dans la partie gérant la logique du jeu, on trouve cette ligne

🐍 **Script Python**

```
if KeysPressed[pygame.K_UP] and KeyUp == 0:
    pass
```

La variable `KeyUp` stocke l'état de la touche `[Flèche Haut]` lors de l'affichage précédent. Dans cette condition, on détecte si le joueur vient d'appuyer sur cette touche. Pour l'instant cette condition ne déclenche rien mais cela va changer par la suite.

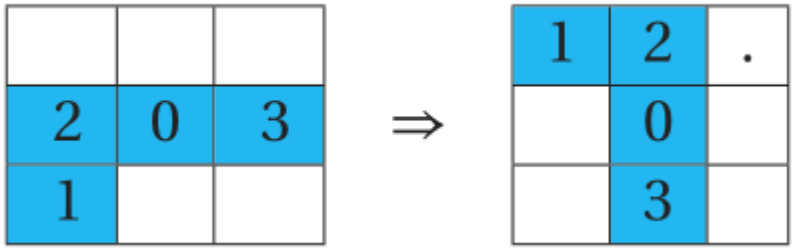
Gestion de la rotation

Vous allez gérer la rotation de la pièce courante. Tout d'abord après la condition gérant l'appui sur la touche `[Flèche Haut]`, vous allez modifier la valeur de la variable `rot`. Chaque appui doit augmenter la variable `rot` de 1. Il serait judicieux d'appliquer un modulo 4 pour faire en sorte que cette variable ne puisse prendre que des valeurs entre 0 et 3. Dans le jeu orignal, les pièces ne tournent que dans un sens.

Question 1 **Question 2** **Question 3**

Créez une fonction `Rot90Droite(P)` qui, à partir d'une pièce 3 \(\times\) 3 tourne cette pièce de \(90^{\circ}\). La pièce P correspond à une liste de listes, cette pièce ne doit pas être modifiée. Vous allez construire une nouvelle pièce et la retourner. Voici quelques conseils :

- Pour créer une nouvelle pièce, vous pouvez l'initialiser à partir d'une liste de listes contenant des 0 ou appliquer la fonction `copy.deepcopy()` sur la pièce P actuelle. Le contenu n'a pas d'importance, car de toute façon, il va être écrasé
- Il faut programmer la rotation de \(90^{\circ}\). Voici un exemple avec la pièce P en entrée et la pièce R à calculer à droite.
  Dans tous les cas, la case centrale ne change pas.



- Option 1 : écrivez une instruction pour chacune des huit cases. Par exemple, pour la case 1 en haut à gauche : `R[0][0]=P[2][0]`, et pour la case 2 : `R[0][1]=P[1][0]`.
- Option 2 : faites une liste des positions des huit cases des bords, ceci en tournant dans le sens des aiguilles d'une montre : `L=((0,0) , (1,0) , (2,0), (2,1) ( (2,2) , (1,2) ...]` Ainsi en créant une boucle for d'indice i allant de 0 à 7 , vous savez que la case à la position `R[i]` doit être initialisée avec la case `L[(i-2)%8]`.

Créez une fonction `Rotn(P,nb)` qui calcul la pièce P après nb rotations.
Pour cela :

- Initialiser une pièce 3 \(\times\) 3 sous forme de liste de listes. Il est judicieux d'utiliser la fonction `copy.deepcopy()` pour cloner la pièce P, car si la variable nb vaut 0, il n'y aura aucune rotation effectuée et c'est la copie de la pièce initiale qui sera retournée.
- Effectuez autant de rotations que nécessaire. Pour cela utiliser la fonction `Rot90Droite()`.
- Retournez le résultat.

Modifier la fonction `AffPiece()` pour qu'elle tienne compte de la variable rot et affichez la pièce en tenant compte de ce paramètre. Maintenant, lorsque vous appuyer sur la touche [Flèche Haut], vous devez voir la pièce tourner.

Déplacement latéraux

Question 1 **Question 2**

Écrivez une fonction `DetectColission()` qui détermine suivant une pièce, une rotation et une position (x,y) données s'il y a collision avec le décor ou non. Voici quelques conseils :

- Appliquer la rotation sur la pièce pour obtenir sa bonne orientation
- Créez une double boucle d'indices dx et dy pour parcourir les cases de la pièce.
- Comparer chaque case `(dx,dy)` de la pièce avec la case `(x+dx,y+dy)` du décor. Si les deux cases sont non vides, alors il y a collision, et retournez vrai dans ce cas.

Complétez le code gérant l'appui sur les touches `[Flèche Droite]` et `[Flèche Gauche]`. Par exemple, lors de l'appui sur `[Flèche`

`Gauche]`, vérifiez d'abord que la futur place de la pièce n'est pas en collision avec le décor. Si aucune collision n'est détectée, alors modifier la position de la pièce en faisant : \(px-=1\).

Gestion de la descente

Question 1 | Question 2 | Question 3 | Question 4 | Question 5

Écrivez une fonction `FusionDecor()` qui, suivant une pièce, une rotation et une position (x,y) donnée, fixe cette pièce dans le décor. Cette fonction est comparable à la fonction `DetectCollision()`, sauf qu'il n'y a pas à faire de test, mais juste un transfert des cases colorées de la pièce vers les cases de la grille.

Écrivez une fonction `NextPiece()` qui initialise une nouvelle pièce. Pour cela, grâce au package `random`, choisissez une pièce au hasard. Sa position sera forcément la ligne 0 et au milieu de la grille, c'est-à-dire à l'abscisse 6. Par contre vous pouvez choisir sa rotation aléatoirement.

Tous les 20 affichages, la pièce courante descend automatiquement d'une ligne, gérez la collision avec le décor. Lorsque la pièce est susceptible de descendre, examinez si sa position futur produit une collision. Dans ce cas-là, elle ne doit pas descendre, car elle est stoppée par quelque chose. Appelez cette fonction `FusionDecor()` pour figer la pièce. Après cela générez une nouvelle pièce.

Vous pouvez maintenant gérer l'appui sur la touche `[Flèche BAS]`. Le mécanisme est identique à celui de la descente automatique.

Il reste un mécanisme à mettre en place : le retrait des lignes pleines. Cet événement peut arriver après la fusion d'une pièce avec le décor. Il se peut qu'une ou plusieurs lignes pleines apparaissent. Écrivez une fonction `RetraitLigne()` dont l'objectif est de retirer l'ensemble des lignes pleines du décor.

- Créez une boucle for avec un indice partant de 0 jusqu'à 15 compris. Les deux dernières lignes ne doivent pas être traitées.
- Faites un calcul pour trouver la valeur de l'indice qui parcourt les lignes en sens inverse, c'est-à-dire de l'indice 15 à 0.
- Testez la ligne associée à ce nouvel indice pour savoir si elle est pleine :
    - Pour cela, il suffit de détecter si une valeur 0 est présente dans la ligne courante. utilisez le test `0 in MaLigneCourante` qui retourne Vrai ou Faux
    - Si la ligne est pleine, retirez-la grâce à la

fonction `DECOR.pop(index)`.

- Une fois le parcours terminé, des lignes ont pu être supprimées. Ainsi tant que le nombre de lignes dans la liste `DECOR` est insuffisant, rajoutez des lignes vides à l'indice 0 grâce à la fonction `DECOR.insert(0,...)`. Pensez à insérer une ligne vide qui soit indépendante de la constante `LIGNE_VIDE` définie dans le programme.