DA COSTA
Mélissa

TS2

DOSSIER ISN

Image du menu du jeu

Projet: Jeu snake

CHEF Loïc, ZERBIB Kévin

2016

I\ Pourquoi avons-nous choisi ce projet:

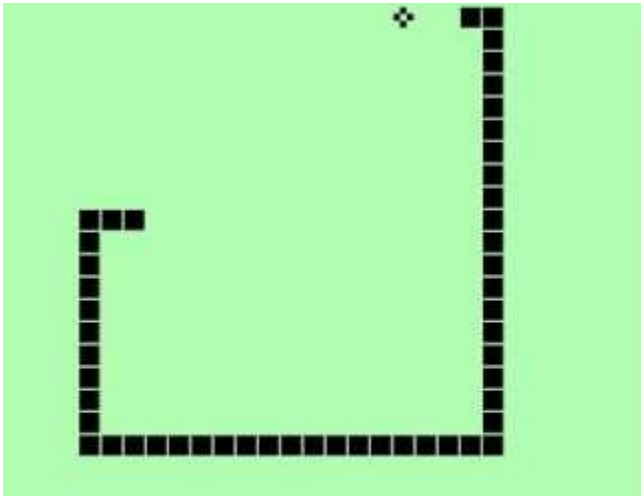
Le choix de ce projet nous est venu très rapidement, réaliser un mini-jeu est très plaisant, c'était ce qui nous attirait le plus. Le jeu snake nous paraissait intéressant à réaliser lorsque l'on observe les différents mécanismes du jeu. De plus il nous paraissait adapté à nos compétences en ISN, et réalisable avec le langage appris: Python.

II\Principe du jeu snake:

Le jeu snake, de l'anglais signifiant «serpent», est un jeu vidéo populaire créé au milieu des années 1970. Il s'est de nouveau fait connaître dans les années 1990 (notamment grâce à Nokia, une entreprise de télécommunications) avec l'émergence du nouveau support de jeu qu'est le téléphone portable. Aujourd'hui, il est toujours aussi populaire et est devenu un classique dans les jeux vidéo.

Le joueur contrôle une longue et fine créature semblable à un serpent, qui doit slalomer entre les bords de l'écran et les obstacles qui parsèment le niveau. Pour gagner chacun des niveaux, le joueur doit faire manger à son serpent un certain nombre de pastilles ou de fruits (de la nourriture en général), allongeant à chaque fois la taille du serpent. Alors que le serpent avance inexorablement, le joueur ne peut que lui indiquer une direction à suivre (en haut, en bas, à gauche, à droite) afin d'éviter que la tête du serpent ne touche les murs ou son propre corps, auquel cas il risque de mourir.

Le niveau de difficulté est contrôlé par l'aspect du niveau (simple ou labyrinthique), le nombre de fruits à manger, l'allongement du serpent et sa vitesse.



Exemple d’un jeu Snake

III\Début du projet:

En commençant nos recherches pour la réalisation du projet ils nous a semblé utile d'installer pygame. Pygame est une bibliothèque libre multiplate-forme qui facilite le développement de jeux vidéo avec le langage de programmation Python.
Ensuite, nous avons cherché sur internet des exemples de codes du jeu snake pour nous rendre compte de l'ampleur du projet. Nous avons réussi à identifier les différentes étapes du programme et à comprendre les bases du code que nous allons devoir effectuer. En observant ces codes, nous avons remarqué que pygame n'était pas utilisé, nous avons alors pris la décision de ne pas utiliser pygame car il n'est pas indispensable, et risquerait de nous compliquer le travail. Ensuite nous avons cherché à modifier un des codes trouvés, celui qui nous paraissait le plus simple, pour mieux comprendre les fonctions utilisées en modifiant certaines données. Malgré cela, nous avons eu du mal à comprendre entièrement le programme. D'une part car ils étaient en anglais ce qui nous a empêché de comprendre entièrement les explications et d'autre part car nos capacités de codage avec python sont réduites. En effet, certaines fonctions utilisées dans le code étaient trop compliquées pour nous, notamment une des fonctions principales celle qui permettait le déplacement du serpent.

A\Répartition des tâches:

Pour sortir de ce problème, nous avons demandé de l'aide à une connaissance qui est étudiant en maths informatique. Il nous à alors expliqué pas à pas la réalisation de notre code en nous indiquant les fonctions qu'ils faillaient utiliser. Loïc et moi nous sommes occupés de la réalisation du nouveau code, nous avons suivi chacun de notre côté les instructions données, puis nous les mettions en commun pour réparer d'éventuelles erreurs.
Ensuite pour rendre notre projet plus agréable, nous avons eu l'idée d'intégrer à notre jeu un menu. Indépendamment à la réalisation du code, Kévin s'est donc occupé de réaliser le menu. Pour cela il l'a codé en utilisant python mais aussi pygame. Il s'est aussi occupé d'ajouter une musique de fond pendant le temps de la partie qui nécessitait aussi pygame.

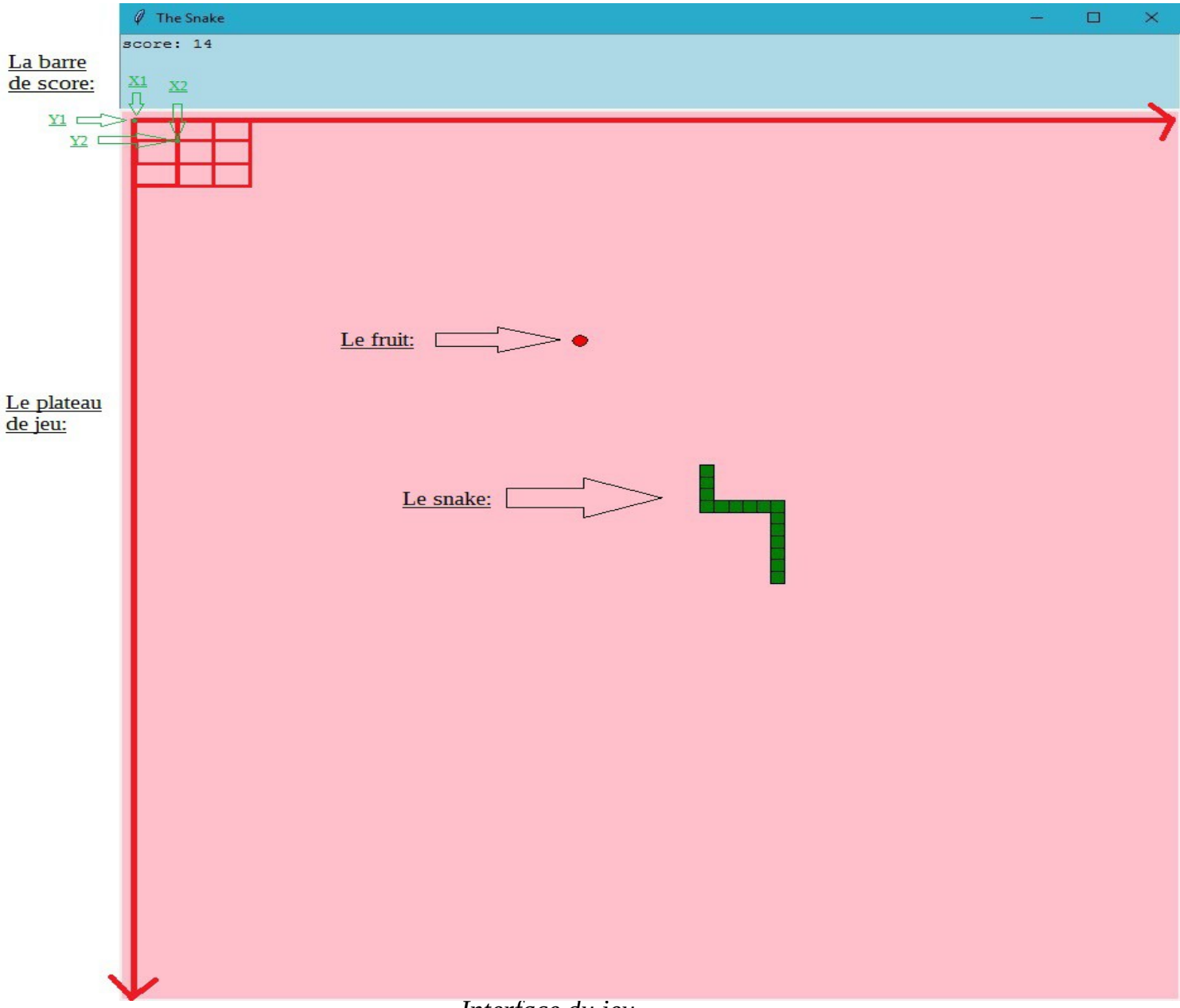
B\Objectifs du projet final:

Pour notre projet final nous souhaitons intégrer une barre de score qui augmenterait lorsque le snake aura mangé un fruit.
Puis nous souhaitons qu'a chaque début de partie le snake apparaisse aléatoirement, ainsi que le fruit. De plus, nous souhaitons que le menu possède un bouton «play» et que la musique de fond se répète jusqu'à la fin de la partie.

IV\Réalisation du projet:

A\Interface de la fenêtre de jeu:

La fenêtre du jeu est composée de deux parties: d'abord la partie la plus petite en haut représente la barre des scores, et la partie la plus grande représente le plateau de jeu.



Interface du jeu

Nous tenions à ce que l'affichage de la fenêtre s'adapte à la taille de l'écran de l'ordinateur. Les fonctions fenetre.winfo_screenheight et fenetre.winfo_screenwidth permettent de récupérer les dimensions de l'écran sur lequel la fenêtre s'ouvre, puis la fonction fenetre.geometry nous à permis de définir la taille de la fenêtre en fonction de la taille de l'écran.

Nous avons créé dans la fenêtre deux parties grâce à la fonction: canvas pour le plateau de jeu et la fonction: text pour pouvoir insérer du texte dans la barre de score. Ces fonctions nous ont permis de choisir leurs tailles.
Nous avons réalisé le plateau de jeu en s'aidant d'un repère orthogonal. Le repère nous a permis de définir les cases du plateau.

B\Les valeurs initiales:

Bien qu'elles soient écrites à la fin du programme, il me semble important pour commencer de décrire les valeurs initiales des variables. En effet, celles-ci nous servirons tout au long du programme et sont la base de certaines fonctions. Toutes ces variables sont des variables dites globales c'est-à-dire qu'elles pourront être modifiées dans la suite du programme avec différentes fonctions.
D'abord, pour symboliser le serpent nous avons créé une liste snake, elle concerne toutes les cases qui vont représenter le serpent au fur et à mesure de la partie. Au début, la case qui symbolise le serpent est une case générée aléatoirement, c'est pourquoi dans notre liste nous avons: [case_aleatoire()]. C'est une fonction qui génère les coordonnées d'une case aléatoirement, nous le verrons plus tard.
Ensuite de la même manière, nous avons la variable fruit, contrairement au snake ce n'est pas une liste car il n'y aura à chaque fois qu' un seul fruit dans la partie. Cette variable est égale à une autre: fruit_aleatoire elle se base sur le même principe que case_aleatoire. Nous avons aussi la variable mouvement qui comme son nom l'indique représente le mouvement du snake. Au début de la partie celui-ci est nul donc cette variable est égale aux coordonnées (0, 0). Puis nous avons la variable perdu, qui est égale à 1 lorsque l'on perd, donc au début elle est égale à 0 car nous n'avons pas encore perdu. Pour finir fenetre.after permet de lancer la fonction: tache qui est la fonction principale du jeu que nous expliquerons plus tard.

C\Créations du serpent et du fruit:

Grâce aux cases du plateau nous avons défini une fonction: case_aleatoire qui génère aléatoirement en début de partie les coordonnées d'une case. A partir de ces coordonnées, la fonction: dessine_serpent créé plusieurs rectangles et les colorient en vert grâce à la fonction: remplir_case afin de symboliser le serpent.
Nous avons utilisé le même principe pour crée le fruit. Avec les fonctions: fruit_aleatoire et dessine_fruit. Mais pour générer un fruit, il faut veillez à ce que celui-ci n'apparaisse pas à l'intérieur du serpent, car le but est d'aller chercher le fruit, pas qu'il nous tombe directement dessus. C'est pourquoi nous avons ajouté une fonction: etre_dans_snake(case) qui permet avec la boucle if de déterminer si les coordonnées de la case où va se trouver le fruit sont dans le snake ou pas, else. A partir de là une boucle while permet de dessiner aléatoirement un nouveau fruit, à chaque fois que le précédent vient d'être mangé, à condition qu'il ne soit pas dans le serpent.

D\Déplacement du serpent:

Le déplacement du serpent se fait avec les quatre touches directionnels du clavier. Il nous à donc fallu créer quatre fonctions différentes pour chacune de ces touches. Grâce au repère de notre plateau de jeu nous avons associé à chacune des flèches une direction. Chaque direction est représenté par la même variable global appelé mouvement qui se modifiera automatiquement pour chaque direction voulue. Cette direction est sous forme de coordonnées. Par exemple, pour aller vers la gauche, la fonction: left_key(event) est associé au mouvement (-1,0). Si on lit les coordonnées, cela signifie que le serpent se déplace d'une case vers la gauche. Pour se déplacer vers le haut, la fonction: up_key(event) est associé aux coordonnées (0,-1). Le serpent se déplacera d'une case vers le haut, le signe moins est présent car notre repère est situé en haut à gauche du plateau de jeu. C'est le même principe pour les deux autres directions.

Maintenant il faut que ces fonctions se lance lorsque l'on appuie sur les touches. Suite à une pression sur la flèche de gauche, la fonction qui lui est associé se lance grâce à: fenetre.bind("<Left>", left_key). Chaque flèches directionnels à une fonction qui permet de détecter lorsque l'on appuie sur l'une d'entre elles et de lancer directement le mouvement qui lui correspond.

E\Agrandissement du serpent:

La fonction: mise_a_jour_snake nous permet de gérer l'agrandissement du serpent lorsque l'on mange un fruit. Pour cela on doit modifier deux variables global crée précédemment: la liste snake et la variable fruit. D'abord, on doit récupérer les coordonnées de la dernière case ajoutée au serpent à l'instant où il récupère le fruit. On associe alors ces coordonnées que l'on appelle: AncienneTeteX, et AncienneTeteY à la liste: snake[0]. Le «0» correspond à la dernière case du serpent. Ensuite on récupère la direction du mouvement réalisé par le joueur au même instant en associant à la variable mouvement les coordonnées MouvementX et MouvementY. Grâce à ces coordonnées on peut définir la position de la nouvelle tête du serpent qui sera donc égale aux coordonnés de: (AncienneTeteX + MouvementX, AncienneTeteY + MouvementY)
Ensuite grâce à une boucle if, on détermine lorsque le serpent mange un fruit, c'est-à-dire lorsque les coordonnées de la nouvelle tête sont égaux aux coordonnées de la variable fruit. De là on génère un nouveau fruit et on lance la fonction: mise_a_jour_score qui va alors augmenté le score de la partie de 1 point.
La deuxième partie de la boucle: else agit lorsque l'on ne mange pas de fruit, on enlève alors le dernier élément de la liste snake avec la fonction: snake.pop , c'est-à-dire que l'on ne grandit pas.

F\Fin de la partie:

La partie est finie lorsque le joueur meurt. Le joueur est considéré comme mort à plusieurs conditions: lorsque le serpent se mort la queue et lorsqu'il dépasse les bords de la fenêtre. Nous définissons alors une nouvelle fonction: serpent_mort (NouvelleTete). NouvelleTete représente les coordonnées données au serpent après sa mort. Lorsque les coordonnées du serpent sont supérieur au nombre de cases du plateau, cela signifie que le serpent à dépassé les bords de la fenêtre. Ensuite grâce à la fonction: etre_dans_snake, que nous avons crée précédemment, nous pouvons déterminer lorsque le serpent est dans lui-même. C'est-à-dire lorsqu'il se mange. De là, une boucle if permet de détecter ces deux conditions et d'associer à la variable perdu le chiffre 1, ce qui signifie que le joueur est mort.

De là, on réinitialise alors le jeu grâce à la fonction: reinitialiser_jeu. Cette fonction va permettre de rétablie les valeurs initiales de toutes les variables du programme expliqué dans la première partie.
Aussi, nous avons défini la fonction tache qui est la fonction principale du jeu. Elle se lance dès le début de la partie dans les fonctions initiales et permet de lancer toutes les autres fonctions. Elle met à jour l'affichage et les événements du clavier, indispensable pour les mouvements, grâce à fenetre.update et fenetre.update_idletasks. Puis elle supprime tous les éléments du plateau grâce à Plateau.delete("all") pour les lancer les fonctions: mise_a_jour_snake et dessine_fruit, ainsi que dessine_serpent, expliquées avant.

Ensuite nous avons créé une boucle if qui permet de gérer la fin de la partie. Si le joueur est mort, la fonction tache efface la barre des score affiche au joueur qu'il à perdu et prépare la nouvelle partie grâce à la fonction: reinitialiser_jeu. Aussi elle relance cette fonction principale: tache avec un délais de 70 millisecondes.
Si le joueur n'a pas perdu, la fonction principale tache ne fais que se relancer.

V\Conclusion:

A\Améliorations possibles du projet:

Mes idées pour améliorer le programme sont d'ajouter la possibilité de mettre le jeu en «pause», ce qui apporterait un confort en plus pour le joueur. Aussi, nous pourrions augmenter la difficulté du jeu et le faire varier en ajoutant des obstacles au joueur. Comme par exemple des fruit qui apporteraient des malus. Pour aller encore plus loin, nous pourrions changer le fond du jeu et accélérer la vitesse du serpent au bout d'un certain score.

B\Ce que m'a apporter la réalisation du projet:

Durant cette année en ISN j’ai appris les bases de la programmation. Notamment à coder un site internet en HTML ainsi qu’un jeu en python. A travers le codage du jeu j’ai beaucoup appris, j'ai appris à réaliser quelque chose de plus concret dans un langage informatique et j'ai su trouvé des solutions à des problèmes. Ce projet d’ISN était plaisant à faire et m’a donné goût à la programmation. Ceci m’a motivé à poursuivre mes études vers un DUT Informatique.

ANNEXE: CODE DU JEU:

```
#On importe le module tkinter pour l'interface graphique
from tkinter import *
# pour l'aléatoire
from random import randint
#On importe pygame pour la musique et le menu
import pygame
from pygame.locals import*

###############################################################################
# Création de la fenêtre du menu
FenetreMenu = Tk()

# on donne un nom à cette fenêtre
FenetreMenu.title("Menu du Snake")

# création du canvas
canvas = Canvas(FenetreMenu, width = 1500, height = 450, background = "black")
photo = PhotoImage(file="logo 1.png")
canvas.create_image(0, 0, anchor=NW, image=photo)
# on place le canvas
canvas.pack()

# on crée les boutons du menu
Button(FenetreMenu, text ="Play", fg="yellow", bg="grey", command = FenetreMenu.destroy, relief =
FLAT).pack(side = TOP, padx = 10, pady = 10)
# la boucle de la fenêtre du menu s'arrêtera lorsqu'on appuiera sur la croix rouge
FenetreMenu.mainloop()

# création surface d'affichage
FenetreMenu = pygame.display.set_mode((300,300))
# on initialise le module mixer
pygame.mixer.init()

# le programme va chercher dans les fichiers le son format WAV
# on créer l'objet sound à l'aide du module mixer de pygame
son = pygame.mixer.Sound("Most_awesome_8-bit_song_ever.wav")
# le son se lance avec ses paramètres
son.play(loops = -1, maxtime = 0, fade_ms= 0)

###############################################################################
#On créer une fenêtre grâce à la fonction Tk()
fenetre = Tk()

#On change le titre de la fenêtre
fenetre.title('The Snake')

###############################################################################
#On récupère les dimensions de l'écran
hauteur = fenetre.winfo_screenheight()
```

```
largeur = fenetre.winfo_screenwidth()

#On convertie les données de la hauteur (H) et de la largeur (L) en int, puis en string,
#et on modifie les dimensions voulues
H = str(int(hauteur/1.1))
L = str(int(largeur/2))

#On applique la taille voulue à la fenêtre grâce à l'appel de fonction geometry(wxh+X+Y) où:
# w, h sont respectivement la largeur et la hauteur
# X et Y sont les coordonnées du point d'origine relativement au coin haut-gauche de la fenêtre
# le x et les deux + sont des séparateurs
# pour insérer les variables, on les concatène avec des chaînes de caractères au moyen de l'opérateur de concaténation +
fenetre.geometry(L + "x" + H + "+0+0")

################################################################################################
#On définition des dimensions du plateau de jeu
LargeurPlateau = largeur /2
HauteurPlateau = hauteur /1.2

#On crée un Canvas pour le plateau de jeu
Plateau = Canvas(fenetre, width = LargeurPlateau, height = HauteurPlateau, bg = "pink")
#"side" désigne l'endroit où débute le canvas
Plateau.pack(side="bottom")

#On crée un Canvas pour le score
Barre = Text(fenetre, width = int(largeur /2), height = int(HauteurPlateau / 10), bg = "light blue")
#On place la barre
Barre.pack(side="top")
#On écrit le score initial dans la barre
Barre.insert(END, "score: 0\n")

#On défini le nombre de cases du plateau
NombreDeCases= 75

#On défini les dimenssions d'une case
LargeurCase = (LargeurPlateau / NombreDeCases)
HauteurCase = (HauteurPlateau / NombreDeCases)

################################################################################################
#Fonction qui détermine la taille des cases du plateau et qui les colore en vert pour symboliser le serpent
def remplir_case (x, y):

    #On défini les coordonnées (origine_caseX1; origine_caseY1) du point en haut à gauche de la case
    #et (origine_caseX2;origine_caseY2) du point en bas à droite de la case
    OrigineCaseX1 = x * LargeurCase
    OrigineCaseY1 = y * HauteurCase
    OrigineCaseX2 = OrigineCaseX1 + LargeurCase
    OrigineCaseY2 = OrigineCaseY1 + HauteurCase

    #remplissage du rectangle
```

```
    Plateau.create_rectangle(OrigineCaseX1, OrigineCaseY1, OrigineCaseX2, OrigineCaseY2,
fill="green")

#On renvoie une case aléatoire
def case_aleatoire():

    AleatoireX = randint(0, NombreDeCases - 1)
    AleatoireY = randint(0, NombreDeCases - 1)

    return (AleatoireX, AleatoireY)

# affichee le serpent, l'argument étant la liste snake
def dessine_serpent(snake):

    #tant qu'il y a des cases dans snake
    for case in snake:

        # on récupère les coordonées de la case
        x, y = case
        # on colorie la case
        remplir_case(x, y)

#############################################################################################
#On retourne le chiffre 1 si la case est dans le snake, 0 sinon
def etre_dans_snake(case):

    if case in SNAKE:
        EtreDedans = 1
    else:
        EtreDedans = 0

    return EtreDedans

#On renvoie un fruit aléatoire qui n'est pas dans le serpent
def fruit_aleatoire():

    # choix d'un fruit aléatoire
    FruitAleatoire = case_aleatoire()

    # tant que le fruit aléatoire est dans le serpent
    while (etre_dans_snake(FruitAleatoire)):
        # on prend un nouveau fruit aléatoire
        FruitAleatoire = case_aleatoire

    return FruitAleatoire

#On dessine le fruit, idem que pour colorier une case, mais on utilise create_oval à la place
def dessine_fruit():

    global FRUIT

    x, y = FRUIT
```

```
    OrigineCaseX1 = x * LargeurCase
    OrigineCaseY1 = y * HauteurCase
    OrigineCaseX2 = OrigineCaseX1 + LargeurCase
    OrigineCaseY2 = OrigineCaseY1 + HauteurCase

    #On remplie l'ovale en rouge pour le fruit

    Plateau.create_oval(OrigineCaseX1, OrigineCaseY1, OrigineCaseX2, OrigineCaseY2, fill = "red")

################################################################################
#Ces quatres fonctions permettent le déplacement dans quatres directions du serpent
#elles mettent à jour les coordonées du mouvement
def left_key(event):
    global MOUVEMENT
    MOUVEMENT = (-1, 0)

def right_key(event):
    global MOUVEMENT
    MOUVEMENT = (1, 0)

def up_key(event):
    global MOUVEMENT
    MOUVEMENT = (0, -1)

def down_key(event):
    global MOUVEMENT
    MOUVEMENT = (0, 1)

# indique les fonctions à appeler suite à une pression sur les flèches (ne fonctionne que si la fenêtre a le focus)
fenetre.bind("<Left>", left_key)
fenetre.bind("<Right>", right_key)
fenetre.bind("<Up>", up_key)
fenetre.bind("<Down>", down_key)

################################################################################
# met à jour la variable PERDU indiquant si on a perdu
def serpent_mort(NouvelleTete):

    global PERDU

    NouvelleTeteX, NouvelleTeteY = NouvelleTete

    # si le serpent se mange lui-même (sauf au démarrage, c'est-à-dire: sauf quand MOUVEMENT vaut (0, 0))
    # OU si on sort du canvas
    if (etre_dans_snake(NouvelleTete) and MOUVEMENT != (0, 0)) or NouvelleTeteX < 0 or NouvelleTeteY < 0 or NouvelleTeteX >= NombreDeCases or NouvelleTeteY >= NombreDeCases:
        # alors, on a perdu
        PERDU = 1
```

```
# met à jour le score
def mise_a_jour_score():

    global SCORE

    SCORE = SCORE + 1
    Barre.delete(0.0, 3.0)
    Barre.insert(END, "score: " + str(SCORE) + "\n")

# met à jour le snake
def mise_a_jour_snake():

    global SNAKE, FRUIT

    # on récupère les coordonées de la tête actuelle
    (AncienneTeteX, AncienneTeteY) = SNAKE[0]
    # on récupère les valeurs du mouvement
    MouvementX, MouvementY = MOUVEMENT
    # on calcule les coordonées de la nouvelle tête
    NouvelleTete = (AncienneTeteX + MouvementX, AncienneTeteY + MouvementY)
    # on vérifie si on a perdu
    serpent_mort(NouvelleTete)
    # on ajoute la nouvelle tête
    SNAKE.insert(0, NouvelleTete)

    # si on mange un fruit
    if NouvelleTete == FRUIT:
        # on génère un nouveau fruit
        FRUIT = fruit_aleatoire()
        # on met à jour le score
        mise_a_jour_score()
    # sinon
    else:
        # on enlève le dernier élément du serpent (c'est-à-dire: on ne grandit pas)
        SNAKE.pop()

################################################################################
# réinitialise les variables pour une nouvelle partie
def reinitialiser_jeu():

    global SNAKE, FRUIT, MOUVEMENT, SCORE, PERDU

    # serpent initial
    SNAKE = [case_aleatoire()]
    # fruit initial
    FRUIT = fruit_aleatoire()
    # mouvement initial
    MOUVEMENT = (0,0)
    # score initial
    SCORE = 0
    # variable perdu initiale (sera mise à1 si le joueur perd)
    PERDU = 0
```

```
# fonction principale
def tache():

    # on met à jour l'affichage et les événements du clavier
    fenetre.update
    fenetre.update_idletasks()
    # on met à jour le snake
    mise_a_jour_snake()
    # on supprime tous les éléments du plateau
    Plateau.delete("all")
    # on redessine le fruit
    dessine_fruit()
    # on redessine le serpent
    dessine_serpent(SNAKE)

    # si on a perdu
    if PERDU:
        # on efface la barre des scores
        Barre.delete(0.0, 3.0)
        # on affiche perdu
        Barre.insert(END, "Perdu avec un score de " + str(SCORE))
        # on prépare la nouvelle partie
        reinitialiser_jeu()
        # on rappelle la fonction principale
        fenetre.after(70, tache)
    # sinon
    else:
        # on rappelle la fonction principale
        fenetre.after(70, tache)

################################################################################
# le snake initial: une liste avec une case aléatoire
SNAKE = [case_aleatoire()]
# le fruit initial
FRUIT = fruit_aleatoire()
# le mouvement initial, une paire d'entiers représentant les coordonées du déplacement, au départ on ne
bouge pas
MOUVEMENT = (0, 0)
# le score initial
SCORE = 0
# la variable permettant de savoir si on a perdu, sera mise à 1 si on perd
PERDU = 0

# on appellera la fonction principale pour la première fois juste après être entré dans la boucle de la
fenêtre
fenetre.after(0, tache())

#On crée une boucle qui va afficher la fenêtre
#tant que l'utilisateur ne clique pas sur la croix rouge en haut à droite
fenetre.mainloop()
```