

Snake très simple avec Pyxel, pas à pas

Objectifs

L'objectif de ce petit projet initial est de vous familiariser avec la création de jeux vidéos à l'aide du moteur de jeux « rétro » Pyxel, logiciel libre et open-source.

On utilisera la programmation orientée objet pour créer notre jeu.

On définira donc des **attributs** et des **méthodes**.

Pyxel est un module utilisable via Pyzo en l'installant avec la commande

>> ***pip install pyxel***


On l'utilise ensuite en important le module au début du script : ***import pyxel***



Cahier des charges :

- le serpent se meut automatiquement, on peut le déplacer avec les flèches du clavier.
- s'il mange la pomme, il grandit et celle-ci réapparaît dans une case vide
- s'il quitte l'écran ou se mord, il meurt, et le jeu s'arrête.

Principes généraux des jeux vidéos

Un jeu vidéo peut être résumé ainsi : 

Dans Pyxel, la boucle infinie est implicite, et l'attente des quelques millisecondes déjà prise en charge => pas besoin de s'en occuper.

Des méthodes prédéfinies gèrent les actions 2 et 3

Une **boucle infinie** fait progresser le jeu :

A chaque tour :

1. On **écoute les interactions** du joueur
2. On **met à jour** l'état du jeu
3. On **dessine** les éléments à l'écran,
4. On attend quelques millisecondes

action

Mettre à jour l'état du jeu

Dessiner les éléments à l'écran

méthode de la classe *SnakeGame*

update()

draw()

Dans le **constructeur de la classe**, on crée la fenêtre du jeu: `pyxel.init(400, 400, title="snake")`

A la fin du programme, dans le `__main__`, on lance l'exécution du jeu avec `pyxel.run(game.update, game.draw)` après avoir créé un objet `game`. La méthode `run` de `pyxel` fait appel aux **deux méthodes prédéfinies**, qui seront appelées **20 fois par seconde**.

Il existe de nombreuses méthodes toutes faites permettant de dessiner, écrire du texte... Les couleurs sont désignées par des entiers de 0 à 15 (0 désignant le noir.)

Effacer l'écran et le remplir de noir	<code>pyxel.cls(0)</code>
Détection d'interactions utilisateurs	Flèches clavier <code>pyxel.btn(pyxel.KEY_RIGHT)</code> ou UP, LEFT, DOWN barre espace : <code>pyxel.btn(pyxel.KEY_SPACE)</code>
Ecrire du texte	<code>pyxel.text(50,64, 'GAME OVER', 7)</code>
Dessiner un rectangle	<code>pyxel.rect(x, y, long, larg, 1)</code> x et y : coords du sommet haut gauche. Ensuite les dimensions. Dernier paramètre : la couleur

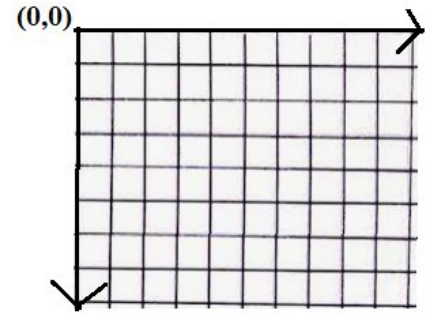
Version 1 : dessiner le serpent

La grille

Les cases seront représentées par des coordonnées. L'origine est en haut à gauche. On commence à zéro, la 1^{ère} coordonnée est l'abscisse (numéro de colonne) et la seconde l'ordonnée (numéro de ligne)

Exemple : ici, la grille a pour dimensions 200x160 pixels, et **10 cases par 8**. Chaque case est carrée de côté pixels.

On définit alors les attributs HEIGHT, WIDTH, CASE (en majuscules car ce sont des constantes : convention !)



Puis on peut créer la fenêtre avec `pyxel.init`

Coordonnées de la case en bas à gauche ; et en haut à droite

```
def __init__(self):
    # constantes de jeu (ne sont pas modifiées)
    self.TITLE = "snake"
    self.WIDTH = 200
    self.HEIGHT = 160
    self.CASE = 20
    self.FRAME_REFRESH = 10 # Gère la vitesse de jeu

    # initialiser l'application Pyxel
    pyxel.init(self.WIDTH, self.HEIGHT, title=self.TITLE)
```

Le serpent

Le serpent est représenté par un **attribut double liste** : `self.snake = [[3, 3], [2, 3], [1, 3]]`, définie dans le **constructeur** (après `pyxel.init`)

Le premier élément est sa **tête**, elle est en **[3,3]** ensuite vient son **corps**.

Représenter le serpent initial sur la grille ci-dessus. Quel est son nombre d'anneaux initial ?

Dessiner le serpent

Pour dessiner sur l'écran les cases du serpent, on utilise la méthode `pyxel.rect(x,y,L,l,color)`

- `x` et `y` sont les coordonnées du coin supérieur gauche, `L` et `l` les dimensions du rectangle.
- `color` est un indice entre 0 et 15 désignant une couleur de la palette prédéfinie Pyxel.

Les instructions suivantes seront placées dans la méthode `draw()`

```
# dessiner le corps en vert
for anneau in self.snake[1:]:
    x, y = anneau[0], anneau[1]
    pyxel.rect(x * self.CASE, y * self.CASE, self.CASE, self.CASE, 11)

# dessiner la tête en orange
x_head, y_head = self.snake[0]
# 9 est la couleur orange
pyxel.rect(x_head * self.CASE, y_head * self.CASE, self.CASE, self.CASE, 9)
```

Ecrire le score

Au début, l'attribut `score` vaut 0 (à définir au même endroit que l'attribut `snake`, au niveau principal du programme, à l'extérieur de toute méthode). On le mettra à jour plus tard dans la méthode `update()`, mais on peut déjà écrire le score initial sur la fenêtre, par une instruction dans la méthode `draw()`

```
#le score
#7 est la couleur blanche
pyxel.text(4, 4, f"SCORE : {score}", 7)
```

Enfin, on écrit une méthode `update()` pour l'instant vide, et on lance le jeu avec `pyxel.run`

Jalon 1
Le serpent
est dessiné

Version 2 : animer le serpent

Jusqu'ici, le serpent ne bougeait pas. On va l'animer un peu.

Déplacer le serpent « tout droit »

Pour commencer, on va supposer que la **direction de déplacement** du serpent est `self.direction = [1,0]` c'est-à-dire que le serpent va On ajoute l'attribut `direction` dans le **constructeur**.

```
# la nouvelle tête est l'ancienne, déplacée dans la direction
self.head = [self.snake[0][0] + self.direction[0], self.snake[0][1] + self.direction[1]]
# on l'insère au début
self.snake.insert(0, self.head)
```

Exemple : au début, on a `self.snake = [[3, 3], [2, 3], [1, 3]]`

Que devient maintenant l'attribut `self.snake` après un déplacement ?

Que dire de la taille du serpent ?

On efface le dernier élément de `self.snake` pour terminer le mouvement : `self.snake.pop()`

A FAIRE : Intégrer ces instructions dans la méthode `update()` qui est appelée automatiquement par Pyxel 30 fois par seconde, et lancer le programme.

Que se passe-t-il ?

Ralentir le jeu

30 images par secondes (*ou Frames Per Second FPS*), ça donne une bonne fluidité d'affichage, mais ça fait quand même trop rapide pour le mouvement du serpent. Pour ralentir, on va utiliser le compteur de frames intégré à Pyxel, en effectuant le mouvement par exemple uniquement tous les 15 frames.

On rajoute la constante `self.FRAME_REFRESH = 15` au début du constructeur, puis dans la méthode `update` on met le mouvement au sein d'un test. Vérifiez : le mouvement est beaucoup plus lent !

```
if pyxel.frame_count % self.FRAME_REFRESH == 0:
    # la nouvelle tête est l'ancienne, déplacée dans la direction
    self.head = [self.snake[0][0] + self.direction[0], self.snake[0][1] + self.direction[1]]
    # on l'insère au début
    self.snake.insert(0, self.head)
    self.snake.pop()
```

Changer la direction du serpent

Cela va se faire dans la méthode `update()` en « écoutant » les interactions du joueur (quand il tape sur une touche du clavier) avec `pyxel.btn`

```
# En cas d'appui sur une touche : Changer direction du snake
if pyxel.btn(pyxel.KEY_ESCAPE):
    exit()
elif pyxel.btn(pyxel.KEY_RIGHT) and self.direction in ([0, 1], [0, -1]):
    self.direction = [1, 0]
elif pyxel.btn(pyxel.KEY_LEFT) and self.direction in ([0, 1], [0, -1]):
    self.direction = [-1, 0]
```

Jalon 2
Le serpent
doit tourner

Question : à quoi sert la première ligne dans le if ?

Version 3 : faire mourir le serpent

Dans notre version du jeu : le serpent meurt lorsqu'il se mord la queue, ou lorsqu'il quitte l'écran. Dans ce cas, le jeu s'arrête, et on quitte la fenêtre.

Pour savoir si la tête du serpent a touché son corps : on teste si les coordonnées de la tête correspondent à un anneau déjà existant du serpent :

Pour savoir si la tête du serpent « sort » de la fenêtre : on doit vérifier plusieurs conditions :

- En abscisse : c'est dedans si ET
donc ça sort si OU
- En ordonnée : c'est dedans si ET
donc ça sort si OU

```
# Faire mourir le snake s'il se touche lui-même
if self.head in self.snake[1:] \
    or self.head[0] < 0 \
    or self.head[0] > self.WIDTH/self.CASE - 1 \
    or self.head[1] < 0 \
    or self.head[1] > self.HEIGHT/self.CASE - 1:
    exit()
```

D'où la condition multiple dans la méthode `update()`

Vérifiez son fonctionnement : pour le cas où il se mord la queue, vous aurez besoin de définir au début un serpent plus long. On pourra remplacer `exit()` par `pygame.quit()`

Jalon 3
Le serpent
peut mourir

Version 4 : manger la pomme... et réagir !

On place une pomme (matérialisée par une case rose) au hasard dans la fenêtre. Lorsque le serpent mange la pomme, il grandit d'un anneau (sa queue n'est pas effacée), et le score augmente de 1.

Attribut représentant la pomme :

```
self.food = [8, 3]
# dessiner la nourriture
x_food, y_food = self.food
# 8 est la couleur rose
pygame.rect(x_food * self.CASE, y_food * self.CASE, self.CASE, self.CASE, 8)
```

(au début, on la place arbitrairement)

Comment tester si le serpent a mangé la pomme ? On teste si les coordonnées de la tête coïncident avec celles de la pomme, donc si

Pour replacer une nouvelle pomme, on tire au hasard des coordonnées dans la grille. Pour cela, on a besoin de la fonction **randint**.

randint doit être importée depuis le module **random**, au tout début du programme

```
import pygame
from random import randint
```

`randint(a,b)` renvoie un entier aléatoire entre a et b.

On recommence jusqu'à ce que ces coordonnées soient OK (pas « dans le corps du serpent »)

```
while self.food in self.snake:
    self.food = [randint(0, self.WIDTH/self.CASE - 1), \
                 randint(0, self.HEIGHT/self.CASE - 1)]
# sortie du while : on a trouvé une nouvelle case pour la pomme
```

Jalon 4
Le serpent grandit, la
pomme réapparaît

Extensions possibles

A ce stade, le jeu est terminé ! Plusieurs améliorations sont possibles : au lieu de quitter si le serpent meurt, relancer instantanément une nouvelle partie ; conserver un high score (tant qu'on ne quitte pas le jeu puis de manière persistante en l'écrivant dans un fichier), améliorer les graphismes, ajouter du son...