Une initiation à la bibliothèque Pyxel

Pyxel est un moteur de jeu vidéo rétro pour Python.
Grâce à ses spécifications simples inspirées par les consoles rétro, comme le fait que seulement 16 couleurs peuvent être affichées et que seulement 4 sons peuvent être lus en même temps, vous pouvez vous sentir libre de créer des jeux vidéo dans le style pixel art.
Les spécifications et les API de Pyxel sont inspirées de PICO-8 et TIC-80.
Pyxel est un logiciel libre et open source. Lors de la nuit du code, nous vous fournirons un environnement de jeu (images, sons etc ...), vous aurez alors 6 h pour créer votre jeu à l'aide de la bibliothèque Pyxel.

Pour chaque étape de se tutoriel de découverte, retrouvez sur studserver ce que vous devez obtenir.

Vous êtes bien sur libres de prendre des libertés avec ce tutoriel.
N'hésitez pas à tester, à modifier, à améliorer le jeu que vous allez créer en suivant les étapes de ce tutoriel.
Faîtes preuve d'imagination et de créativité !

Nous allons utiliser l'app en ligne Pyxel Studio. Vous pouvez pour plus de confort installer Pyxel sur votre machine :

Pour travailler vous avez trois possibilités :

Utiliser votre IDE préféré (spyder) en important le module `pyxel` au début de votre programme.
Cette méthode est à privilégier si `pyxel` est installé sur la machine sur laquelle vous travaillez.

Utiliser l'app en ligne Pyxel Studio : https://www.pyxelstudio.net/.

Utiliser le cahier numérique associé à ce tutoriel : www.cahiernum.net/N2RGVP

Copiez le lien indiqué en haut de la page du nouveau projet.
Sauvegardez le en lieu sur (clef USB + mail + agenda ....).
Il vous sera nécessaire pour retrouver votre travail et ne pas recommencer à zéro à chaque fois.

Pyxel est normalement installé sur les machines de la salle E215.
Pour installer Pyxel sur votre ordinateur :
https://github.com/kitao/pyxel/blob/main/docs/README.fr.md#comment-installer

Étape 1 : Créer une application avec Pyxel

Un jeu vidéo peut se résumer de la façon suivante :

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 attend quelques millisecondes.

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

Des fonctions prédéfinies gèrent les actions 2 et 3 :

Action	fonction Pyxel prédéfinie
Mettre à jour l'état du jeu	`update()`
Dessiner les éléments à l'écran	`draw()`

Au début du programme, on crée la fenêtre du jeu : `pyxel.init(128, 128)`
A la fin du programme, on lance l'exécution du jeu avec `pyxel.run(update, draw)` qui fait appel aux deux fonctions prédéfinies, qui seront appelées 20 fois par seconde.

```
import pyxel

pyxel.init(128, 128) #les dimensions de la fenetre de jeu

carre_x = 60
carre_y = 60

def update():
        if pyxel.btnp(pyxel.KEY_Q):
                pyxel.quit()

def draw():
        pyxel.cls(0) #efface la fenetre
        #creation d'un rectangle
        #dont le coin en haut a gauche est positionne
        #en coordonnees (carre_x, carre_y)
        #8 , 8 sont les dimensions du rectangle et 11 sa couleur (0-15)
        pyxel.rect(carre_x, carre_y, 8, 8, 11)

pyxel.run(update, draw)
```

Dans ce premier exemple, le rectangle est immobile car la fonction `draw()` effectue toujours la même suite d'instructions.
La fonction `update`, permet d'écouter les interactions de l'utilisateur. Ici elle fermera l'application lorsque la touche `Q` sera appuyée.

Pour déplacer le carré en utilisant la flèche droite du clavier, on peu créer une fonction `deplacement` que l'on va appeler dans la fonction `update`.

```
import pyxel

pyxel.init(128, 128) #les dimensions de la fenetre de jeu
carre_x = 60
carre_y = 60

def deplacement(x, y):
    """deplacement a l'aide de la fleche droite du clavier"""

    if pyxel.btn(pyxel.KEY_RIGHT) :
        x = x + 1

    return x, y

def update():
    global carre_x, carre_y

    if pyxel.btn(pyxel.KEY_Q):
        pyxel.quit()

    #mise a jour de la position du carre
    carre_x, carre_y = deplacement(carre_x, carre_y)

def draw():
    pyxel.cls(0)
    pyxel.rect(carre_x, carre_y, 8, 8, 11)

pyxel.run(update, draw)
```

A vous de jouer

Compléter la fonction `deplacement` pour pouvoir déplacer votre carré dans toutes les directions (attention, il ne doit pas pouvoir sortir de la fenêtre de jeu)

Vous pouvez retrouver la liste des touches du clavier sur le lien suivant : https://github.com/kitao/pyxel/blob/main/python/pyxel/__init__.pyi

Vous trouverez sur studserver une démo de ce à quoi pourrait ressembler votre jeu à la fin de cette étape.

Étape 2 : Quand notre carré devient une arme de destruction massive

A chaque appui sur la barre d'espace, notre carré doit lancer un missile, on ajoute pour cela une variable de type `list` qui va stocker les tirs créés (chaque tir est un tuple : ses coordonnées) ainsi qu'une fonction `tirs_creation`.

En début de code :

```
tirs_liste = []
```

La fonction pour créer un tir avec la touche espace :

```
def tirs_creation(x, y, tirs_liste) :
    # btnr pour eviter les tirs multiples
    if pyxel.btnr(pyxel.KEY_SPACE):
        tirs_liste.append([x+4, y-4])
    return tirs_liste
```

La fonction `update` :

```
def update():
    global carre_x, carre_y, tirs_liste

    #mise a jour de la position du carre
    carre_x, carre_y = deplacement(carre_x, carre_y)

    # creation des tirs en fonction de la position du vaisseau
    tirs_liste = tirs_creation(carre_x, carre_y, tirs_liste)
```

et la fonction `draw` :

```
def draw():
    pyxel.cls(0)
    pyxel.rect(carre_x, carre_y, 8, 8, 11)

    # tirs
    for tir in tirs_liste:
        pyxel.rect(tir[0], tir[1], 1, 4, 10)
```

A vous de jouer

Bon évidemment avec tout ça les tirs restent statiques ....
Il suffit de créer une fonction `tirs_deplacement(tirs_liste)` pour qu'à chaque mise à jour de la fenêtre, les tirs se déplacent d'un pixel vers le haut (et disparaissent lorsqu'ils sortent de la fenêtre de jeu).

Modifiez la fonction `update()` pour que ces déplacements soient effectifs.

Vous trouverez sur studserver une démo de ce à quoi pourrait ressembler votre jeu à la fin de cette étape.

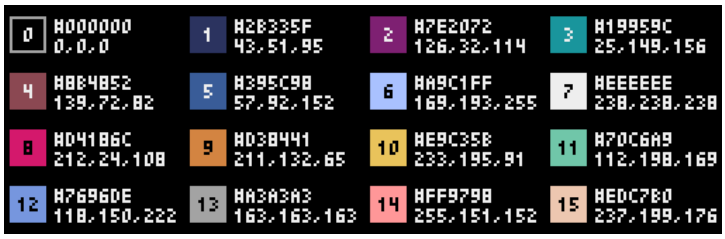
Étape 3 : Ajouter des ennemis

Les ennemis seront stockés dans une liste (comme pour les tirs).
A sa création un ennemi sera créé sur une position aléatoire en haut de l'écran (utilisation de la fonction `randint` du module `random`) puis à chaque frame descendra d'un pixel vers le bas.
Bien entendu lorsqu'un ennemi sort du cadre il doit être supprimé de la liste.

Pour créer un ennemi chaque seconde, on peut utiliser le code suivant :

```
if (pyxel.frame_count%30==0): #30 frame par seconde
        ennemis_liste.append([random.randint(0, 120), 0])
```

Les ennemis seront dessinés par un carré de dimensions 8x8 en utilisant une couleur non utilisée (les couleurs sont définies par un entier entre 0 et 15 (le vaisseau possède la couleur 11 et les tirs la couleur 10).



Retrouvez la palette de couleurs en ligne :
https://raw.githubusercontent.com/kitao/pyxel/main/docs/images/05_color_palette.png.

A vous de jouer

Modifier votre code pour obtenir le résultat escompté.

Vous trouverez sur studserver une démo de ce à quoi pourrait ressembler votre jeu à la fin de cette étape.

Étape 4 : où l'on rend nos tirs efficaces et notre carré vulnérable

Pour faire disparaître les ennemis lorsqu'ils sont touchés par nos tirs, il suffit d'ajouter une fonction `ennemis_suppression` :

```
def ennemis_suppression():
    """disparition d un ennemi et d un tir si contact"""

    for ennemi in ennemis_liste:
        for tir in tirs_liste:
            if ((ennemi[0] >= tir[0] and ennemi[0] <= tir[0] + 8 ) or \
             (ennemi[0] + 8 <= tir[0] + 8  and \
             ennemi[0]+8 >= tir[0])) and ennemi[1]+9 >= tir[1]:

                ennemis_liste.remove(ennemi)
                tirs_liste.remove(tir)
```

et d'appeler cette fonction dans la fonction `update` :

```
        # suppression des ennemis et tirs si contact
        ennemis_suppression()
```

A vous de jouer

Modifiez votre code pour que lorsque votre carré est touché 3 fois par un ennemi, le jeu s'arrête avec le message "game over" affiché sur l'écran.
Lorsqu'un ennemi touche le carré, il doit disparaître.

On pourra utiliser une variable `vie`, initialisée à 3 et qui est diminuée de 1 lorsque notre carré rencontre un ennemi.

Pour afficher un message, il suffit d'utiliser la syntaxe :
`pyxel.text(50,64, "GAME OVER", 7)`

Vous trouverez sur studserver une démo de ce à quoi pourrait ressembler votre jeu à la fin de cette étape.

Étape 5 : des explosions

L'objectif de cette partie est d'ajouter des explosions lorsque votre carré touche un ennemi ou lorsqu'un de vos tirs touche un ennemi.

Les explosions seront implémentées par une liste contenant leur coordonnées, et une valeur initiale de 12 qui à chaque frame sera décrémentée de 1.
L'explosion changera de couleur et de taille à chaque frame, et disparaîtra au bout de 12 frames.
Il faudra donc stocker les explosions en cours dans une liste `explosions_liste`.
Il vous faudra ajouter une fonction `explosions_creation` et une fonction `explosions_animation` :

```
def explosions_creation(x, y):
    """explosions aux points de collision entre deux objets"""
    explosions_liste.append([x, y, 12])


def explosions_animation():
    """animation des explosions"""
    for explosion in explosions_liste:
        explosion[2] =  explosion[2] - 1
        if explosion[2] == 0:
            explosions_liste.remove(explosion)
```

A vous de jouer

A vous de modifier les fonctions `update` et `draw` (et éventuellement `ennemis_suppression`) pour que cela fonctionne.

Pour dessiner les cercles, on utilisera la syntaxe :
`pyxel.circb(x, y, rayon, couleur)`

Il vous faudra également afficher le nombre de vies restantes au joueur dans la fenêtre (initialement 3 par exemple).

Vous trouverez sur studserver une démo de ce à quoi pourrait ressembler votre jeu à la fin de cette étape.

Étape 6 : De toute façon je ne combat des aliens que dans un vaisseau digne de ce nom. Pas dans un carré !

Il ne nous reste plus qu'à afficher un vrai vaisseau et de vrai ennemis à la place de nos carrés.

- Télécharger l'environnement de jeu `NDC1.pyxres` prévu pour cet exemple sur studserver.

- – Si vous utilisez l'app Pyxel Studio ou le cahier numérique :
    * Menu Burger : 
    * uploader le fichier NDC1.pyxres : 
    * Pour éditer le fichier : 
  – Si vous utilisez un IDE avec Pyxel installé sur votre machine :
    * Le fichier ressource doit être situé dans le même répertoire que votre fichier python.
    * Dans un terminal, déplacez vous dans ce répertoire (commande `cd`).
    * La commande `pyxel edit NDC1.pyxres` permet d'ouvrir le fichier dans l'éditeur de ressources.

Pour utiliser cette ressource, il faut placer à la racine du code la ligne :

```
pyxel.load('NDC1.pyxres')
```

juste après la ligne

```
pyxel.init (128 , 128)}
```

Pour placer une image à l'écran on utilisera la syntaxe :

```
pyxel.blt(x, y, img, u, v, w, h, c)}
```

Cette instruction permet de placer à la position `(x, y)` la région dont le coin supérieur gauche est en `(u, v)` dans la banque d'images `img` et de dimensions `w, h`.
Si une valeur négative est mise pour `w` (ou `h`), la copie sera inversée horizontalement (ou verticalement). Si `c` est spécifiée, elle sera traitée comme une couleur transparente.

Par exemple pour remplacer notre carré par un vrai vaisseau, on écrira dans la fonction `draw()` :

```
pyxel.blt(carre_x, carre_y, 0, 0, 0, 8, 8, 0)
```

A vous de jouer

Utilisez l'environnement de jeu pour dessiner de vrais ennemis et de vrais missiles.

Vous trouverez sur studserver une démo de ce à quoi pourrait ressembler votre jeu à la fin de cette étape.

Améliorations possibles

Vous avez maintenant un jeu tout à fait fonctionnel.
Néanmoins rien ne vous interdit de faire preuve d'imagination pour l'améliorer :

- ajouter un compteur qui comptabilise le nombre d'ennemis abattus ;
- Chaque alien qui arrive à passer au travers de vos tirs devraient vous faire perdre des points de vie ;
- Augmenter la vitesse de déplacement des aliens au fur et à mesure du déroulement du jeu ;
- Les aliens peuvent eux aussi tirer des missiles ;
- ...
  Bref, toute amélioration qui vous passe par la tête sera la bienvenue !

Vous trouverez toute la documentation de pyxel en suivant ce lien :
https://github.com/kitao/pyxel/blob/main//docs/README.fr.md

Ce tutoriel est inspiré de https://nuitducode.github.io/DOCUMENTATION/PYTHON/01-presentation/