Projet NSI

Création d'un jeu

L'environnement de travail

On fera l'ensemble du travail avec Thonny (ou Spyder).
On peut se procurer Thonny à cette adresse (https://thonny.org/). On peut également installer EDUPYTER(https://www.edupyter.net/) qui contient Thonny et un jupyter installable sur clé USB.
La bibliothèque que nous utiliserons est `pyxel`, elle permet de mettre en place un rendu graphique très proche des applications de l'informatique naissante des années 80...

Installation :
Si ce n'est pas déjà fait, il faut installer la bibliothèque `pyxel`.
Il faut écrire la commande suivante :

```
pip install pyxel
```

- Dans la console de spyder.
- Dans outils/ouvrir la console systeme pour thonny.

Vérifions que tout s'est bien passé...
Écrire le code suivant qui doit produire une fenêtre de 128×128 pixels:

```
import pyxel

# taille de la fenetre 128x128 pixels
# ne pas modifier
pyxel.init(128, 128, title="TP_pyxel_0")

def update():
    """mise à jour des variables (30 fois par seconde)"""
    pass

def draw():
    """cette fonction s'exécute 30 fois par seconde"""
    # vide la fenetre
    pyxel.cls(0)

pyxel.run(update,draw)
```

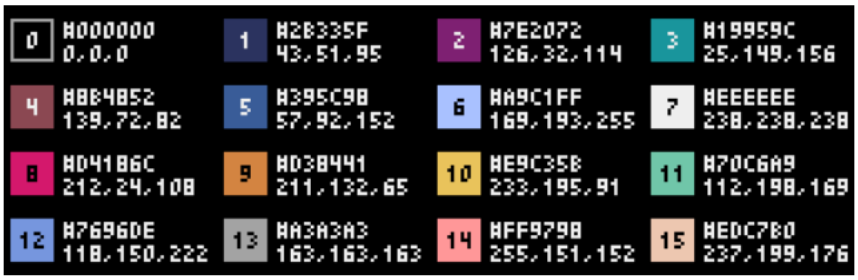
dans spyder, si vous ne voyez pas la fenêtre, il faut aller dans outils/préférences/ipython console/-graphics/ et : (mettre le backend à automatic)

Partie 1 - Un carré que l'on déplace avec les flèches dans la fenêtre

Vous compléterez au fur et à mesure votre code.
Pour placer un rectangle dans la fenêtre, in faut donner ses coordonnées, ses dimensions et sa couleur avec l'instruction : `pyxel.rect(abscisse,ordonnée, longueur, largeur, couleur)`.
Nous nommerons l'abscisse : `vaisseau_x` et l'ordonnée : `vaisseau_y`, pour ses dimensions on souhaite faire afficher un carré de 8×8 pixels et enfin pour les couleurs il faut choisir un

nombre entre 0 et 15 (voici la palette des couleurs disponibles).

Palette de couleurs



Les coordonnées de notre vaisseau vont changer, il faudra les déclarer comme variables globales, cela se fait en début de programme. Le coin supérieur gauche de la fenêtre a pour coordonnées (0,0) et donc le coin inférieur droit a pour coordonnées (126,126).
le vaisseau est un carré dont le coin supérieur droit a pour coordonnées `(vaisseau_x,vaisseau_y)`.
voici le code :

```
import pyxel

# taille de la fenetre 128x128 pixels
# ne pas modifier
pyxel.init(128, 128, title="TP_pyxel_0")
# position initiale du vaisseau
# (origine des positions : coin haut gauche)
vaisseau_x = 60
vaisseau_y = 60

def update():
    """mise à jour des variables (30 fois par seconde)"""
    pass

def draw():
    """cette fonction s'exécute 30 fois par seconde"""
    # vide la fenetre
    pyxel.cls(0)
    # vaisseau (carre 8x8 et couleur 2)
    pyxel.rect(vaisseau_x, vaisseau_y, 8, 8, 2)


pyxel.run(update,draw)
```

Cela produit une fenêtre avec un carré aux coordonnées (60,60)

Nous allons maintenant faire bouger ce carré.

Il nous faudra créer une fonction `vaisseau déplacement(x,y)` qui prend en paramètre les coordonnées du vaisseau et qui les modifient en fonction des flèches du clavier. Puis nous mettrons à jour les coordonnées du vaisseau dans la fonction. `update`

À FAIRE 1:

Compléter la fonction `vaisseau_deplacement(x, y)`, on se déplacera d'un pixel.

```
def vaisseau_deplacement(x, y):
    """déplacement avec les touches de directions"""

    if pyxel.btn(pyxel.KEY_RIGHT):
        if (x < 120) :
            x = x + 1
    if pyxel.btn(pyxel.KEY_LEFT):
        if (x > 0) :
            x = x ...
    if pyxel.btn(pyxel.KEY_DOWN):
        if (y < 120) :
            y = y ...
    if pyxel.btn(pyxel.KEY_UP):
        if (y > 0) :
            y = y ...
    return x, y
```

Modifier la fonction `update`

```
def update():
    """mise à jour des variables (30 fois par seconde)"""
    global vaisseau_x, vaisseau_y
    # mise à jour de la position du vaisseau
    vaisseau_x, vaisseau_y = vaisseau_deplacement(vaisseau_x, vaisseau_y)
```

Nous avons maintenant une fenêtre avec un carré qui se déplace...

Partie 2 - On ajoute des tirs...

On crée une liste dans laquelle on ajoutera les coordonnées du tir à chaque fois que l'on appuiera sur la barre d'espace. Puis on mettra en mouvement ces tirs en modifiant ses coordonnées.
Les tirs auront pour origine le centre du vaisseau.

À FAIRE 2:

On ajoute en début de programme la liste `tirs_liste` et on crée la fonction `tirs_creation`

```
import pyxel
# taille de la fenetre 128x128 pixels
# ne pas modifier
pyxel.init(128, 128, title="TP_pyxel_2")
# position initiale du vaisseau
# (origine des positions : coin haut gauche)
vaisseau_x = 60
vaisseau_y = 60
# initialisation des tirs
tirs_liste = []
```

La fonction `tirs_creation` qui prend en paramètre la position du carré et la liste des tirs.

```
def tirs_creation(x, y, tirs_liste):
    """création d'un tir avec la barre d'espace"""
    # btnr pour eviter les tirs multiples
    if pyxel.btnr(pyxel.KEY_SPACE):
        tirs_liste.append([x+4, y-4])
    return tirs_liste
```

Expliquer l'instruction `tirs_liste.append([x+4, y-4])` et pourquoi x+4 et y-4.

.....
.....
.....

Expliquer pourquoi on a choisi d'ajouter les coordonnées sous forme de liste plutôt que sous forme de tuples.

.....
.....

La fonction `tirs_deplacement`

```
def tirs_deplacement(tirs_liste):
    """déplacement des tirs vers le haut et suppression
    s'ils sortent du cadre"""

    for tir in tirs_liste:
        tir[1] -= 1
        if  tir[1]<-8:
            tirs_liste.remove(tir)
    return tirs_liste
```

Expliquer ce que fait la boucle **for** dans cette fonction

.....
.....
.....
.....

Modifier la fonction `update` en rajoutant :

```
    global vaisseau_x, vaisseau_y, tirs_liste

    # creation des tirs en fonction de la position du vaisseau
    tirs_liste = tirs_creation(vaisseau_x, vaisseau_y, tirs_liste)

    # mise a jour des positions des tirs
    tirs_liste = tirs_deplacement(tirs_liste)
```

Expliquer pourquoi l'ordre de ces deux dernières instructions est important.

.....
.....
.....

Modifier la fonction `draw` en rajoutant :

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

Nous avons maintenant une fenêtre avec un carré qui se déplace avec les flèches du clavier et qui lance des tirs avec la touche espace. On avance...

Partie 3 - On ajoute des ennemis...

Le principe est le même que pour les tirs. On initialise une liste `ennemis_liste` en début de programme, on crée deux fonctions `ennemis_creation` et `ennemis_deplacement` et enfin on modifie les fonctions `update` et `draw`.

À FAIRE 3:

Créer une liste `ennemis_liste` vide en début de programme.
Importer la bibliothèque `random` (`import random`).
Voici la fonction `ennemis_creation` qui crée un ennemis toutes les secondes (car le draw s'exécute 30 fois par seconde) dont les coordonnées seront aléatoires sur l'axe des abscisses et égale à 0 sur l'axe des ordonnées.

```
def ennemis_creation(ennemis_liste):
    """création aléatoire des ennemis"""

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

Pourquoi choisir pour abscisse un entier entre 0 et 120 et pas entre 0 et 128?

.....
.....

Créer la fonction `ennemis_deplacement` qui prend en paramètre la liste `ennemis_liste` en suivant cet algorithme:

Données : ennemis_liste
Pour chaque ennemi dans ennemis_liste **faire**
    on incrémente de 1 l'ordonnée de ennemi
    Si l'ordonnée de ennemi est supérieure à 128
     alors
        On élimine l'ennemi
renvoyer ennemis_liste

Pour éliminer un élément d'une liste : `la_liste.remove(l'élément)`.
Ajouter et modifier dans le `update`

```
    global vaisseau_x, vaisseau_y, tirs_liste, ennemis_liste
    # creation des ennemis
    ennemis_liste = ennemis_creation(ennemis_liste)
    # mise a jour des positions des ennemis
    ennemis_liste = ennemis_deplacement(ennemis_liste)
```

Et dans le `draw`

```
    # ennemis
    for ennemi in ennemis_liste:
        pyxel.rect(ennemi[0], ennemi[1], 8, 8, 8)
```

On a maintenant une fenêtre avec un carré se déplaçant en tirant et des ennemis qui descendent dans la fenêtre. Cela se précise....

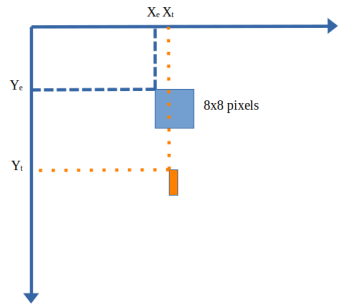
Partie 4 - Gestions des collisions

On commencera par initialiser en début de programme une variable `vies` à 1.
À chaque fois qu'un tir entrera en collision avec un ennemi celui-ci disparaîtra.
Si un ennemi entre en collision avec le vaisseau la variable vie sera mise à 0 et le jeu s'arrêtera.

À FAIRE 4:

Initialiser la variable `vies` à 1 en début de programme.
Il nous faut maintenant créer deux fonctions `ennemis_suppression()` et `vaisseau_suppression()`

La fonction `ennemis_suppression()`
On rappelle que la liste `ennemis_liste` contient les coordonnées de chaque ennemis et que la liste `tirs_liste` contient les coordonnées de chaque tirs.



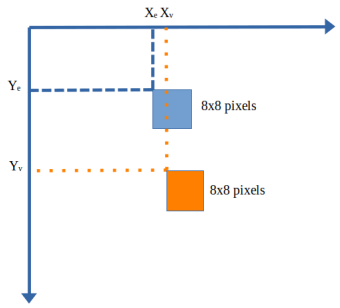
(X_e,Y_e et (X_t,Y_t) sont respectivement les coordonnées de l'ennemi et du tir
On en déduit qu'il y a collision si :

- $X_e \leq X_t \leq X_e + 8$
- $Y_t \leq Y_e + 8$

Implémenter la fonction `ennemis_suppression()` en suivant cet algorithme.

Pour chaque ennemi dans ennemis_liste **faire**
    Pour chaque tir dans tirs_liste **faire**
        Si On détecte la collision **alors**
            On élimine l'ennemi
            On élimine le tir

La fonction `vaisseau_suppression(vies)`.
Cette fonction prendra en paramètre la variable vies et renverra la valeur de cette variable après détection de la collision entre le vaisseau et un ennemi.



Écrire les conditions sur les coordonnées qui permettent la détection d'une collision entre le vaisseau et un ennemi.

.....

.....

.....

.....

Implémenter la fonction `vaisseau_suppression(vie)` en suivant cet algorithme.

Données : vies
Pour chaque ennemi dans ennemis_liste **faire**
    Si On détecte la collision **alors**
        On élimine l'ennemi
        On met le variable vies à 0
renvoyer vies

Mise à jour des autres fonctions

Voici les fonctions `update` et `draw` mise à jour.

```
def update():
    """mise à jour des variables (30 fois par seconde)"""
```

```
    global vaisseau_x, vaisseau_y, tirs_liste, ennemis_liste, vies

    # mise à jour de la position du vaisseau
    vaisseau_x, vaisseau_y = vaisseau_deplacement(vaisseau_x, vaisseau_y)

    # creation des tirs en fonction de la position du vaisseau
    tirs_liste = tirs_creation(vaisseau_x, vaisseau_y, tirs_liste)

    # mise a jour des positions des tirs
    tirs_liste = tirs_deplacement(tirs_liste)

    # creation des ennemis
    ennemis_liste = ennemis_creation(ennemis_liste)

    # mise a jour des positions des ennemis
    ennemis_liste = ennemis_deplacement(ennemis_liste)

    # suppression des ennemis et tirs si contact
    ennemis_suppression()

    # suppression du vaisseau et ennemi si contact
    vies = vaisseau_suppression(vies)
```

```
def draw():
    """création des objets (30 fois par seconde)"""

    # vide la fenetre
    pyxel.cls(0)

    # si le vaisseau possede des vies le jeu continue
    if vies > 0:

        # vaisseau (carre 8x8)
        pyxel.rect(vaisseau_x, vaisseau_y, 8, 8, 1)

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

        # ennemis
        for ennemi in ennemis_liste:
            pyxel.rect(ennemi[0], ennemi[1], 8, 8, 8)

    # sinon: GAME OVER
    else:

        pyxel.text(50,64, 'GAME OVER', 7)
```

Modifier le nombre de vies initial comme vous voulez... ne pas oublier de modifier la diminution du nombre de vies dans la fonction `vaisseau_suppression(vies)`. Et

ajoutez dans le `draw`.

```
        # affichage des vies
        pyxel.text(5,5, 'VIES:'+ str(vies), 7)
```

Cela affichera le nombre de vies dans la fenêtre.

Partie 5 - Simulation d'explosions lors de collisions

Les explosions peuvent être simplement représentées par quelques cercles de rayon croissant (5 par exemple dont le rayon va de 1 à 5 pixels) qui se "propagent" depuis l'ancienne position de l'ennemi.
La gestion des explosions peut se faire sur le même principe que les tirs et les ennemis, à savoir utiliser une liste `explosions_liste`, dans laquelle on ajoutera ou supprimera au besoin des éléments.
Cependant, en plus des coordonnées de "départ" de l'explosion (qui seront bien entendu égales, lors de la création de l'explosion, au milieu de l'ancienne position de l'ennemi), il faut prévoir un troisième élément, qui indiquera à quel "stade" l'explosion est, en stockant par exemple son rayon actuel; la valeur de cet élément sera incrémenté à chaque frame, et lorsqu'il sera devenu égal au rayon du plus grand cercle, l'explosion sera supprimée de la liste.

À FAIRE 5:

Initialiser en début de programme la liste `explosions_liste`.

Voici les deux fonctions à ajouter :

```
def explosions_creation(x, y):
    """explosions aux points de collision entre deux objets"""
    explosions_liste.append([x, y, 0])


def explosions_animation():
    """animation des explosions"""
    for explosion in explosions_liste:
        explosion[2] +=1
        if explosion[2] == 12:
            explosions_liste.remove(explosion)
```

Modification de la fonction `update()` :
Ajouter `explosions_liste` comme variable globale.
Ajouter l'appel de la fonction `explosions_animation()`.
Modification de la fonction `draw()` :

```
# explosions (cercles de plus en plus grands)
for explosion in explosions_liste:
    pyxel.circb(explosion[0]+4, explosion[1]+4, 2*(explosion[2]//4),
    8+explosion[2]%3)
```

Un dernier ajout...
Ajouter-là où il faut l'instruction :`explosions_creation(ennemi[0], ennemi[1])` qui permet l'ajout des coordonnées d'une explosion.

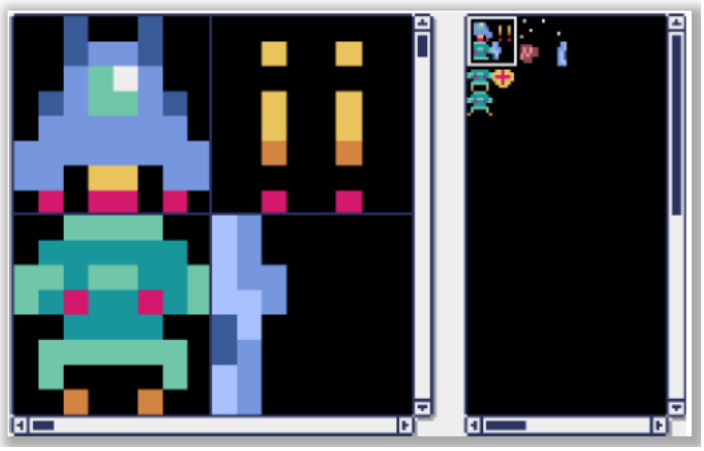
Partie 6 - On utilise des images...

Pyxel permet l'utilisation d'images à la place de simples formes.
Les ressources images se présentent sous la forme d'une banque d'images, c'est à dire un regroupement d'images dans un même ensemble.
Une banque d'images est un ensemble de tuiles (tiles), c'est à dire de petites images (en général) carrées disposées sur une grille de 256 x 256 pixels.
Une tuile peut représenter un sprite ou un élément de décor, avec lequel on peut interagir ou non. L'ensemble complet des tuiles disponibles pour une utilisation dans une zone de jeu est appelé un jeu de tuiles (tileset).
Avec Pyxel, un tileset peut comprendre jusqu'à 3 banques d'images différentes (numérotées de 0 à 2). La première est chargée par défaut dans la banque 0.
Pour cette partie nous utiliserons la banque d'images que vous télé-chargerez à cette adresse
Ce fichier doit être placé dans le même dossier que votre programme.

L'organisation est fondamentale à connaître, car on adresse une image en précisant les coordonnées (x, y) dans la grille de son pixel supérieur gauche, sa taille horizontale et sa taille verticale (en pixels).

Par exemple : la banque d'images dans le fichier images.pyxres contient quelques images, toutes de taille 8 x 8 pixels.

- la première image (un vaisseau) est donc adressée par le pixel (0, 0), et à une taille de 8 pixels horizontaux et 8 pixels verticaux.
- la deuxième (un tir) par le pixel (8, 0) (même taille)
- la troisième par (0, 8) (même taille)
- etc...



À FAIRE 6:

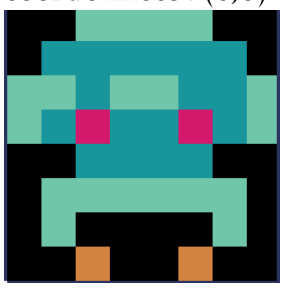
On charge les images dans le programme:
Pour cela il faut écrire l'instruction `pyxel.load("images.pyxres")` en début de programme.

Commençons par le vaisseau.
Pour afficher l'image du vaisseau à la place du carré il faut écrire l'instruction `pyxel.blt(vaisseau_x, vaisseau_y, 0, 0, 0, 8, 8, 0)` à la place de celle qui affiche le rectangle vaisseau.
Explication :
pyxel.blt(x, y, numéro banque, x(image), y(image), taille horizontale, taille verticale, [couleur de transparence])

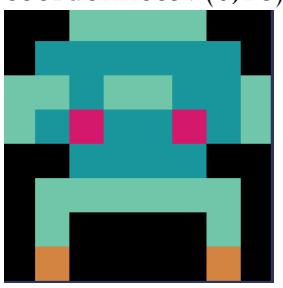
Les ennemis
L'instruction est : `pyxel.blt(ennemi[0], ennemi[1], 0, 0, 8, 8, 8, 0)` à écrire là où il faut.

On peut animer les images, en changeant l'image à afficher toutes les 3 frames par exemple. Dans notre banque d'images, il y a 3 images pour les ennemis :

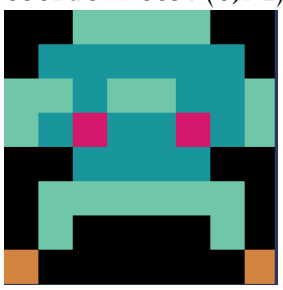
coordonnées : (0,8)



coordonnées : (0,16)



coordonnées : (0,24)



À faire 7:

Voici les instructions pour animer les ennemis :

```
for ennemi in ennemis_liste:
    coeff = pyxel.frame_count % 3 + 1
    pyxel.blt(ennemi[0], ennemi[1], 0, 0, 8*coeff, 8, 8, 0)
```

Faites le nécessaire et expliquer ces instructions.

.....

.....

.....

.....

.....

Il est également possible de modifier/ajouter des images, pour cela il faut les éditer et cela se fait avec une commande en console.

- Dans Thonny : Outils/ouvrir la console du système
- Sinon ouvrir un terminal dans le dossier contenant votre programme et le fichiers images.pyxres

Il faudra écrire la commande : `pyxel edit images.pyxres`, cela ouvre l'éditeur d'images, de sons etc...

Partie 7 - Ajouter un son pour les tirs

Comme pour les images, le son est géré par le fichier `images.pyxres`, dans celui-ci on peut créer des sons et/ou des musiques pour notre jeu..

Il y a déjà un son que l'on peut utiliser pour les tirs, voici le code à ajouter dans le programme là où a lieu la création d'un tir...
speed peut prendre la valeur de 1 à 100 et 1 est la plus rapide.

```
        pyxel.sound(0).speed = 1
        pyxel.play(2, 0)
```

Le projet

Par équipe de 2 ou 3 lancez-vous dans la création d'un casse-briques