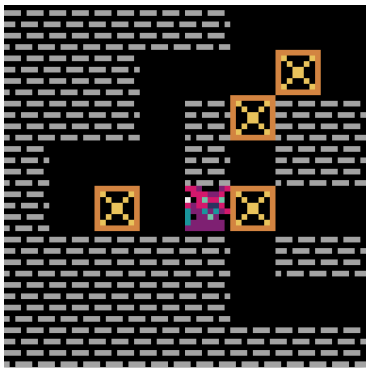
Un Sokoban avec Pyxel

L'objectif de ce tutoriel est d'implémenter un jeu de sokoban avec le module Pyxel.
Principalement vous apprendrez à utiliser les tilesmaps, dans lesquelles les images des banques d'images sont ordonnées en motifs de tuiles.
Vous apprendrez également à gérer les sons dans un jeu 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 !

Comme dans les tutoriels précédents, vous pouvez :

- directement sur votre machine ;
- en utilisant l'app www.cahiernum.net/8RJVWM ;
- en utilisant le cahier numérique

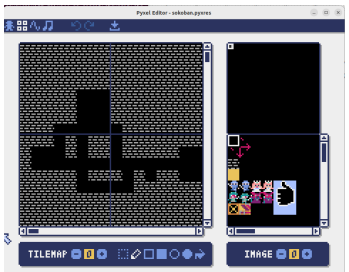
Règles du jeu :
Gardien d'entrepôt (divisé en cases carrées), le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois.
Une fois toutes les caisses rangées (c'est parfois un vrai casse-tête), le niveau est réussi et le joueur passe au niveau suivant, plus difficile en général. L'idéal est de réussir avec le moins de coups possibles (déplacements et poussées). *Source : wikipedia.fr*

Préparation de l'environnement de travail

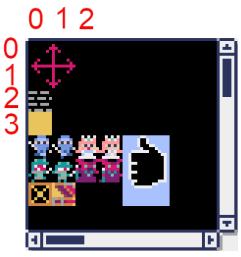
Commencez par télécharger sur studserver les fichiers :

- `sokoban.pyxres`, il contient les éléments du jeu (images et sons) ;
- `sokoban.py`, il contient un début d'implémentation du jeu.

Dans la fonction `draw` on remarquera la ligne `pyxel.bltm(0, 0, 0, 0, 0, 192, 192)` qui mérite quelques explications.
Une tilemap (littéralement une "carte de tuile") est une grille utilisée pour créer la disposition/le fond graphique d'un jeu. L'écran est ainsi représenté par une grille composée de nombreuses cases, sur lesquelles sont appliquées une image par case. Il est alors possible de créer un fond au jeu, composé des éléments du jeu de tuiles.
On accède à l'éditeur de tilemap dans l'éditeur de ressource.
Chaque case du tilemap et donc une image de dimension 8x8 d'une ressource image.



Dans notre exemple, on a rempli les murs par la tuile `(0, 2)` et l'intérieur du labyrinthe par la tuile `(2, 0)`.



La méthode `bltm(x, y, tm, u, v, w, h)` copie la région de taille `(w, h)` de `(u, v)` de la tilemap `tm` (0-7) à `(x, y)`.
Si une valeur négative est mise pour `w` (ou `h`), la copie sera inversée horizontalement (ou verticalement). La taille d'une tuile est 8x8 pixels et elle est storée dans une tilemap en tant que paire `(tile_x, tile_y)`.
Ici `pyxel.bltm(0, 0, 0, 0, 0, 192, 192)` copie donc la région de la tilemap `0` dans le coin haut gauche de la fenêtre. Il s'agit de la région de dimension `(192, 192)` à partir du coin haut gauche de la tilemap.

Comme cette partie de la tilemap est plus grande que les dimensions de la fenêtre, la tilemap ne sera pas affichée dans son intégralité.
La ligne `pyxel.camera(j_x-32, j_y-32)` permet de centrer l'affichage de la tilemap sur les coordonnées du joueur (initialisées à `88`)

Le joueur

4 variables globales sont utilisées pour le personnage du joueur.

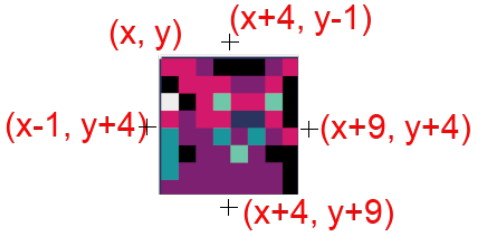
1.
- `j_x` et `j_y` représentent ses coordonnées.
- `costume` est la liste des coordonnées du coin an haut à gauche dans l'image `0` de ses deux costumes.
- `num_costume` est le numéro actuel du costume qui est utilisé pour l'affichage.

Compléter la fonction `draw` pour afficher le joueur.

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.

2. Pour l'instant le joueur ne peut se déplacer que vers le haut. Quelques explications sur l'implémentation de ce déplacement.
La fonction `tuile_haut(x, y)` retourne un couple (tuple) composé d'une valeur égale à 100 par défaut (nous expliquerons le rôle de cette valeur plus tard) et d'un couple indiquant la tuile utilisée dans la tilemap au dessus de la position de notre personnage.
Dans la fonction `update`, la variable `haut` récupère l'évaluation de la fonction `tuile_haut` pour les coordonnées du joueur.
Puis, si la touche `KEY_UP` est activée et si la tuile du dessus est la tuile `(2, 0)` (tuile noire) ou `(0, 3)` (tuile jaune) notre personnage se déplace vers le haut. Ainsi il ne pourra pas passer à travers les murs.

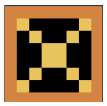
Compléter cette méthode pour pouvoir déplacer le héro dans les quatre directions.



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

Les boites

La liste
`liste_boites = [[40, 56], [56, 48], [56, 56], [40, 40], [16, 80], [40, 80]]`
contient les coordonnées des 6 boites à placer dans le jeu.



Commencez par compléter la fonction `draw` pour dessiner ces boites sur leurs emplacements.

Dans un premier temps, notre joueur ne dois pas pouvoir ce déplacer sur les boites.
En ajoutant les lignes suivantes à la fonction `tuile_gauche(x, y)` celle-ci renverra également le numéro de la boite (0 – 5) située sur la case de coordonnées `(x, y)` et la valeur `100` s'il n'y a pas de boite.

```
for num_boite in range(len(liste_boites)) :
        coord = tuple(liste_boites[num_boite])
        if x-8 == coord[0] and y==coord[1]:
            boite = num_boite
```

Un simple `if gauche[0] == 100` : dans la fonction `update()` permettra d'éviter que le joueur ne se déplace sur une caisse située à sa gauche.

Oui mais il faut aussi pouvoir pousser les caisses !
Compléter la structure conditionnelle pour que si la caisse n'est pas égale à 100 alors le joueur se déplace vers la gauche et la caisse également.

C'est très bien mais on ne doit pas pouvoir pousser la caisse si à gauche de cette dernière se trouve un mur ou une autre caisse. Pour le savoir, il suffit de faire de nouveau appel à la fonction `tuile_gauche` mais avec les coordonnées de la caisse en argument :
`gauche_de_gauche = tuile_gauche(liste_boites[gauche[0]][0], liste_boites[gauche[0]][1])`
La variable `gauche_de_gauche` contiendra un tuple dont la première valeur est le numéro de la caisse située à gauche de la caisse que l'on veut pousser (`100` s'il n'y en a pas) et la seconde la tuile située sur cet emplacement.

Compléter la fonction `update` pour que le joueur puisse déplacer les caisses en respectant les règles du jeu.

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

Fin du jeu

1. Lorsque toutes les cases sont sur la partie jaune. Le jeu doit s'arrêter et l'image du pouce levé doit s'afficher.
   Pour réaliser cela, on pourra créer une fonction `tuile(x, y)` qui retourne la tuile située aux coordonnées `(x, y)`.
   Ensuite, on initialise une variable globale `gagne = False` en début de programme.
   Dans la fonction `update`, on initialise un compteur `cpt = 0`, puis on parcours la `liste_boites` pour connaître le nombre de boites situées sur une tuile jaune.
   Si ce nombre est égale à 6, le jeu s'arrête et le pouce apparaît.

2. En cas d'erreur, le joueur doit pouvoir réinitialiser l'ensemble des variables en utilisant la touche espace et recommencer le niveau.

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

Gestion du son

Il est possible de rajouter des sons et des musiques au jeu. Ces derniers peuvent être crées à partir de l'éditeur de ressource.
Une musique est la superposition de 4 sons simultanément (jusqu'à 4).

La gestion de la musique est déjà (presque) implémentée dans le programme .
Il faut donc ajouter un attribut `self.musique = Bouton_musique()` existe déjà dans la `class Jeu`.
Il suffit alors d'appeler les fonctions `lancerMusique` et `stoperMusqiue` au bon endroit dans la fonction `update`.

Pour les bruitages on va créer des fonctions similaires.
Il faudra prévoir :

- une variable globale `bruitage`
- l'affichage des symboles  lorsque `bruitages` vaut `False` et  sinon.
- Un son particulier lorsque le personnage pousse une caisse et un autre lorsqu'une caisse est positionnée dans la zone jaune.

Pour jouer jouer un petit son on pourra utiliser la syntaxe `pyxel.play(0, 0)`.

Pour améliorer votre jeu, vous pouvez maintenant

- Créer d'autres niveaux (attention, les niveaux doivent avoir une solution).
- Ajouter un chronomètre.
- compter le nombre de déplacements utilisés et stocker les résultats dans un fichier. Le but étant donc alors de résoudre le niveau en le moins de déplacements possibles.
- Ajouter des ennemis.
- de nombreuses améliorations sont possibles, soignez imaginatifs ...