Le labyrinthe infernal

L'objectif de ce troisième tutoriel est d'implémenter un jeu de labyrinthe 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 https://www.pyxelstudio.net/ ;
- en utilisant le cahier numérique https://www.cahiernum.net/SM6HXJ

Préparation de l'environnement de travail

Commencez par télécharger sur studserver les fichiers :

- `laby.pyxres`, il contient les éléments du jeu (images et sons)
- `laby.py`, il contient un début d'implémentation du jeu.
- Si vous utilisez l'app Pyxel Studio :
  - Copier le code `laby.py` dans la console ;
  - Menu Burger :  ;
  - uploader le fichier `laby.pyxres` :  ;

Les différents éléments de jeu sont regroupés en instances de classe. Nous avons donc créé (et commenté) différentes classes :

- La classe `Jeu` qui regroupe les attributs et méthodes du jeu.
  Pour l'instant, il y deux attributs :
  - un `hero` (notre personnage principal) qui est un objet de la classe `Hero`.
  - une liste `mechants` qui est contient des objets de la classe `Mechant`. Pour l'instant cette liste ne contient qu'un unique méchant.

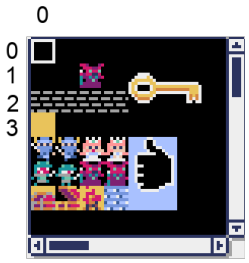
  Dans la fonction `draw` on remarquera la ligne `pyxel.bltm(0, 0, 0, 0, 0, 512, 512)`qui mérite quelques explications.
  Une **tilemap** (littéralement une "carte de tuile") est une grille utilisée pour créer la disposition/le fond graphique d'un jeu. L'écran est ainsi représenté par une grille composée 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.

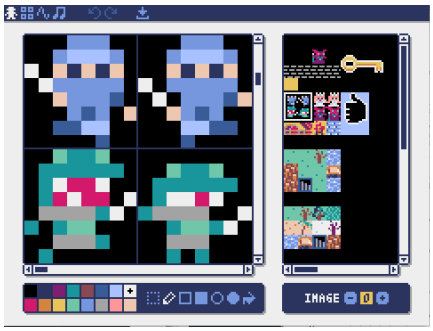
  On accède à l'éditeur de tilemap dans l'éditeur de ressource.

  > Si vous utilisez Pyxel directement installé sur votre machine, pour ouvrir l'éditeur de ressources, dans un terminal déplacez vous jusqu'au répertoire contenant le fichier `laby.pyxres` (commande `cd`) puis `pyxel edit laby.pyxres`.

  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 `(0, 3)`.

  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 stockée dans une tilemap en tant que paire `(tile_x, tile_y)`.
  Ici `pyxel.bltm(0, 0, 0, 0, 0, 512, 512)` copie donc la région de la tilemap `0` dans le coin haut gauche de la fenêtre. Il s'agit de la région de dimension `(512, 512)` 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 classe `Hero` qui va nous permettre de gérer le héro de cette grande aventure.
- La classe `Mechant` qui va nous permettre de gérer les différents ennemis.
- La classe `Coffre` qui va nous permettre au héro d'ouvrir le coffre pour récupérer la clef cachée à l'intérieur.

Le héro

1. Notre héro aura trois attributs :

   - son abscisse `x` ;
   - son ordonnée `y` ;
   - son numéro de costume (soit 0, soit 1).

   La méthode `draw` permet de dessiner le héro avec le bon costume. La variable `costume` sera incrémentée dans la méthode `deplacement`.
   La ligne `if self.costume%10 < 5` permet de changer de costume toutes les 5 incrémentations.
   Ces deux méthodes sont déjà complètes et fonctionnelles.

   > Compléter la méthode `draw` de la classe `Jeu` pour afficher le héro.

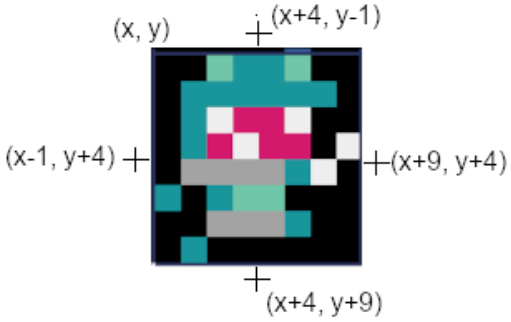
2. La méthode `deplacement` permet de déplacer le héro. Évidemment, il doit rester à l'intérieur du labyrinthe. La fonction

```
def tuile_droite(self):
        return pyxel.tilemap(0).pget((self.x+9)//8, (self.y+4)//8)
```

   retourne la tuile située à la droite du héro.
   Les lignes

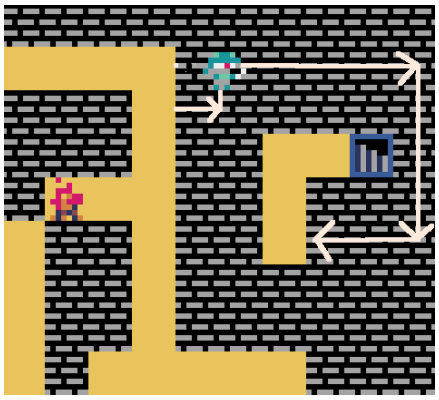
```
if pyxel.btn(pyxel.KEY_RIGHT) and tuile_droite == (0, 3):
            self.x = self.x + 1
            self.costume += 1
```

   permettent "d'écouter" le clavier et de déplacer le personnage vers la droite si la flèche droite du clavier est activée et si la tuile à droite est la tuile `(0, 3)`.

   > Compléter cette méthode pour pouvoir déplacer le héro dans les quatre directions.
   >
   > 
   >
   >
   > Il faudra aussi penser à appeler cette méthode au bon endroit dans votre code.

3. Évidemment lorsque le héro sort de la fenêtre de jeu, il n'est plus possible de la voir.
   Pour remédier à ce problème, il suffit de centrer l'affichage de la tilemap sur les coordonnées du héro en ajoutant la ligne :
   `pyxel.camera(self.hero.x-60, self.hero.y-60)` à la méthode `update` de la classe `Jeu`.

4. Vous avez certainement remarqué une zone inaccessible à droite de la tilemap.
En fait cette zone devrait être accessible à notre personnage s'il en connaît l'accès.



Les tuiles du passage secret ont été remplie par la tuile `(1, 2)` qui a la même apparence que la tuile `(0, 2)`.

Modifiez la méthode `deplacement` pour rendre ce passage secret accessible au personnage.

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

Le coffre

Il vous faut maintenant compléter la méthode `draw` de la classe `Coffre`.

- Si la tuile située à droite du héro est la tuile `(0, 6)` alors vous devez
  - dessiner le coffre ouvert à la position `(224, 56)` ;
  - puis passer l'attribut `ouvert` à `True`.
- Si l'attribut `ouvert` est à `True`, vous devez dessiner une clef en haut à gauche de la fenêtre pour indiquer au joueur qu'il a récupéré la clef.

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

Les méchants

La classe `Mechant` permettant d'implémenter les ennemis est partiellement écrite.
Lors de sa création, un objet de cette classe reçoit en arguments :

- `position` : sa position de départ sous forme de tuple ;
- `costume1` et `costume2` les coordonnées du coin gauche des deux costumes dans la banque d'images 0 servant à l'animation du personnage ;
- `hero` : un objet de la classe `Hero` auquel il sera rattaché (notre personnage principal).‘

Par exemple, dans la méthode `__init__` de la classe Jeu la ligne
`self.mechants = [Mechant((8, 32), (0, 32), (8,32), self.hero)]` permet de créer une liste contenant un seul méchant.

1. Compléter la méthode `draw` de la classe `Mechant` pour que le méchant apparaisse à l'écran.

   Il faudra également ajouter l'appel à la méthode `draw` de classe `Mechant` dans la méthode `draw` de la classe `Jeu`.

2. La méthode `deplacement` de la classe `Mechant` est déjà écrite.

   Vous pouvez la modifier si vous voulez changer le comportement des méchants mais ce n'est pas obligatoire.

   Pour que vos méchants se déplacent, il vous faut compléter la méthode `update` de la classe `Jeu`.

3. .

   Ajouter un méchant (par exemple la princesse... et oui les princesse ne sont pas toutes gentilles) dont la position de départ est `(176, 8)`.
   Ajouter un troisième méchant (le sorcier) avec une position de départ de `(288, 72)`.

4. Pour l'instant vous ne risquez rien car les méchants ne sont pas dangereux.

   Complétez la méthode `touche` de la classe `Mechant`. Cette méthode doit retourner un booléen :
   - `False` si le méchant ne touche pas notre personnage ;
   - `True` dans la cas contraire.

   Vous ajouterez un attribut `envie` à la classe `Jeu` : `self.envie = True`.
   Cet attribut doit passer à `False` si l'un des méchants entre en contact avec le personnage principal.

   Enfin lorsque l'attribut `envie` est à `False`, le jeu s'arrête avec le message "GAME OVER" .
   Il faudra utiliser la syntaxe `pyxel.text(abscisse, ordonnée, texte, couleur)`.

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

Tout est bien qui finit bien

Pour terminer le jeu, il ne vous reste plus qu'à prévoir une fin heureuse.
La partie sera gagnée lorsque que votre personnage rejoindra un des deux escaliers en étant en possession de la clef.

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 implémentée dans le programme grâce à la `class Bouton_musique`.
Il faut donc ajouter un attribut `self.musique = Bouton_musique()` existe déjà dans la `class Jeu`.
Il suffit alors d'appeler la méthode `lancer` de cette `class` sur cet objet après votre initialisation de pyxel puis d'appeler la méthode `maj` dans la méthode `update` ainsi que la méthode `draw`.

Pour les bruitages on va créer une autre `class bouton_bruitages`.
Cette classe doit contenir

- un seul attribut `self.jouer = True` ;
- une méthode `lancer` qui donne la valeur `True` à `jouer` ;
- une méthode `stopper` qui donne la valeur `False` à jouer.
- une méthode `draw` qui affiche  lorsque `jouer` vaut `False` et  sinon.
- une méthode `maj` qui inverse la valeur de jouer lorsque l'icône est cliqué.
- une méthode `bruit` qui permet de jouer un son passé en argument sur la canal 0 lorsque `jouer` vaut `True` :

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

Il ne reste plus qu'à

- ajouter un attribut `bruitage` dans la `class App` et utiliser les méthodes qui lui sont associées lorsqu'un bruitage doit être joué.

Pour améliorer votre jeu, vous pouvez maintenant

- imaginer d'autres niveaux plus compliqués ;
- modifier le déplacements des méchants (niveau facile : des aller retours prévisibles sur les lignes droites, niveau intermédiaire : identique à celui du tutoriel, niveau expert : les méchants se déplacent en direction du héro) ;
- de nombreuses améliorations sont possibles, soignez imaginatifs ...

arb 2024-2025