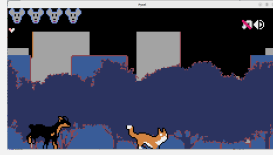


# Un chat, des chiens et des rats



Nous allons utiliser l'app en ligne Pyxel Studio. Vous pouvez pour plus de confort installer Pyxel sur votre machine : <https://github.com/kitao/pyxel/blob/main/docs/README.fr.md#comment-installer>



Ce tutoriel est disponible sous la forme d'un cahier numérique : <https://www.cahiernum.net/VJAQBH>.



Dans un premier temps cliquez sur le bouton **NOUVEAU PROJET**.



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

## Préparation de l'environnement de travail

Commencez par télécharger sur studserver les fichiers :

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

Comme nous l'avons vu avec la simulation sur les lemmings, les différents éléments de jeu sont regroupés en instances de classe. Nous avons donc créé (et commenté) différentes classes :

- La `class App` qui regroupe les attributs et méthodes du jeu ;  
Pour l'instant, il y a trois attributs :
  - un chat (notre personnage principal) qui est un objet de la `class chat` ;
  - un bouton permettant de gérer la musique qui est un objet de la `class bouton_musique` ;
  - un décor qui est un objet de la `class decor`.
- La `class chat` qui va nous permettre de gérer le héros de cette grande aventure ;
- La `class decor` qui va nous permettre de gérer les différents décors ;
- La `class bouton_musique` qui va nous permettre d'activer ou de désactiver la musique.

## Animation du chat



1. Le dessin du chat est effectué grâce à la ligne `self.chat.draw()` dans la méthode `draw(self)` de la class `App`.

Cette ligne permet donc de faire appel à la méthode `draw(self)` de la class `chat`.

Pour animer ce personnage, il va falloir changer le dessin du chat régulièrement.

On crée donc une variable `costume = pyxel.frame_count//3%6`.

Cette variable prendra les valeurs de 0 à 5 en changeant toutes les 20 frames (60//3).

Il suffit alors de choisir le dessin du chat en fonction de la valeur de `costume` :

```
if costume in {0, 1, 2}: # le costume change en fonction de la valeur de coeff
    pyxel.blit(self.x, self.y, 0, 48*costume, 63, 40, 24, 8)
else :
    pyxel.blit(self.x, self.y, 0, 48*(costume - 3), 88, 40, 24, 8)
```

2. Notre chat sait courir, il lui faut maintenant apprendre à sauter.

Dans la class `chat`, nous avons déjà écrit une méthode `up(self)`.

Pour simuler le saut du chat, nous allons utiliser son attribut `vitesse_verticale`.

### A vous de jouer

Appelez la méthode `up` dans la fonction `update` pour que le saut soit actif.

Vous remarquerez qu'une fois dans l'arbre, le chat ne peut plus en descendre. Complétez alors la méthode `up`.



Pour rendre le saut un peu plus réaliste, pensez que dans ce cas, la montée sera plus courte que la descente.



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

## Gestion des décors



1. Pour donner l'impression que le chat se déplace, nous allons faire défiler les arbres de la droite vers la gauche. Rien de plus simple, il vous suffit d'appeler la méthode `maj` de la class `decor` sur l'attribut `decor1` dans la méthode `update`.

Remarquez la présence de la ligne `self.x = self.x - self.vitesse` dans la méthode `maj(self)` de la class `decor`. Cela permet de décrémenter de 1 à chaque frame l'attribut `self.x` qui définit l'abscisse du décor.

Dans la méthode `draw(self)` de cette class nous retrouvons alors la ligne suivante qui nécessite quelques explications :

```
pygame.blit(self.x%460 - 230, self.y, self.num_image, self.x_ressource, self.y_ressource, self.w, self.h, 8)
```

Cela permet de dessiner le décor.

`self.x%460` est le reste de la division euclidienne de `self.x` par 460 c'est à dire un nombre entre 0 et 459 et donc `self.x%460 - 230` permet donc à l'abscisse de l'image de prendre les valeurs de -230 à 229 (c'est à dire que lorsqu'elle arrive à -230, à gauche de la fenêtre, elle repart à 229, à droite de la fenêtre).

Évidemment cela laisse un noir entre deux images des arbres. Pour éviter cela décommentez la ligne suivante.

```
pygame.blit((self.x-230)%460 - 230, self.y, self.num_image, self.x_ressource, self.y_ressource, self.w, self.h, 8)
```

Cela permet de dessiner une autre image des arbres mais en la décalant de 230 pixels par rapport à la première.

### A vous de jouer

Créez un nouvel attribut dans votre class `App` :

```
self.decor2 = Decor(1, 2, 0, 122, 230, 80)
```

 pour faire défiler les immeubles bleus.

Les immeubles doivent être dessinés derrière les arbres.

- 

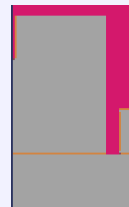


Vous remarquerez que nous avons défini l'attribut `vitesse` de ce décor à 2 tandis que les arbres ont une vitesse de 3. En effet pour obtenir un effet de profondeur, plus un décor est loin moins sa vitesse de défilement doit être grande.



Vous trouverez dans l'image 2 du fichier ressource des silhouettes d'immeubles gris.

- Incorporez les au décor. Ces immeubles doivent défiler plus lentement que les immeubles bleus et doivent être plus haut que les immeubles bleus.



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

## Les chiens et les rats arrivent



### A vous de jouer

Pour implémenter les chiens, nous allons créer une `class Chien`.

Un objet `Chien` possédera deux attributs :

- son abscisse `self.x` initialisée à -50
- son ordonnée `self.y` initialisée à 90 (vous pouvez modifier ces valeurs pour un meilleur rendu)

Cette `class` doit également posséder une méthode `draw(self)` permettant de dessiner et d'animer le chien (inspirez vous de ce qui a été fait pour le chat).



Le chien se déplace plus vite que le chat, l'animation doit donc être plus rapide.

Enfin une méthode `maj(self)` permettra d'incrémenter d'un pixel l'attribut `self.x`.



Vous pourrez tester votre `class` en créant un objet `chien` dans la `class App`.

Pour avoir plusieurs chiens, on va remplacer l'attribut `chien` dans la `class App` par un attribut `chien_liste` contenant une liste de chiens.

```
self.chien_liste = []
```

Puis une méthode `creation_chien(self)` permettra d'ajouter un chien en moyenne toutes les 5 secondes :

```
def creation_chien(self):  
    if (pygame.frame_count % 60 == 0) and randint(0, 4) == 0:  
        self.chien_liste.append(Chien())
```

Une méthode `suppression_chien` permettra de supprimer les chiens qui sortent de la fenêtre de jeu :

```
def suppression_chien(self):  
    for chien in self.chien_liste:  
        if chien.x > 230 :  
            self.chien_liste.remove(chien)
```

Il ne reste plus qu'à mettre à jour dans la méthode `update` l'ensemble des chiens de la liste et dans la méthode `draw` de dessiner tous ces chiens.

### A vous de jouer

Il faut maintenant utiliser la même méthode pour dessiner les rats.

Les rats vont plus lentement que le chat, il vont donc se déplacer de la droite vers la gauche.



Pour ouvrir le fichier de ressource, dans un terminal, déplacez vous jusqu'au répertoire contenant votre projet (commande `cd`) puis commande `pygame edit tuto2.pyxres`  
Vous trouverez sur studserver une démonstration de ce à quoi devrait ressembler votre jeu à la fin de cette étape.



## Le chien mord le chat, le chat mange le rat

Nous allons maintenant compter combien de fois la chat se fait mordre par un chien.

Ajoutez un attribut `self.chien_score = 0` à votre class `App`.

La méthode `disparition_chien` suivante permet à chaque fois qu'un chien touche le chat

- d'enlever le chien de `chien_liste`;
- d'incrémenter `score_chien`.

```
def disparition_chien(self):  
    if self.chat.y > 65 :  
        for chien in self.chien_liste :  
            if chien.x > 60 and chien.x < 110 :  
                self.chien_liste.remove(chien)  
                self.chien_score = self.chien_score + 1
```

L'ajout de la ligne suivante dans la méthode `draw` permet d'afficher le nombre de vie restantes :  
`pyxel.blit(0, 16, 0, 208 + 8*self.chien_score , 56, 8 , 8, 8)`

### A vous de jouer

- Le jeu doit s'arrêter lorsque le chat n'a plus de vie;
- De la même manière lorsque le chat rencontre un rat
  - le rat doit disparaître;
  - A chaque nouveau rat mangé un trophée apparaît en haut de l'écran;
  - Chaque fois que 5 rats réussissent à s'échapper, `chien_score` est incrémenté de 1 (et le chat perd une vie)



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 créés 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`.



Un attribut `self.musique = Bouton_musique()` existe déjà dans votre class `App`.

Il suffit donc 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`.

### A vous de jouer

1. 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 (le dessin se trouve dans l'image 2 du fichier ressource).
- 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` : `pyxel.play(0, 2)` pour jouer le son 2 (cf. fichier ressource).

2. Il ne reste plus qu'à

- ajouter un attribut `bruitage` dans la class `App` et utiliser les méthodes qui lui sont associées ;
- appeler la méthode `bruit` avec le son 5 lorsqu'un chien mord le chat ;
- appeler la méthode `bruit` avec le son 3 lorsque le chat mange un rat.



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

Pour améliorer votre jeu, vous pouvez maintenant

- augmenter la vitesse des chiens et des rats au fur et à mesure du jeu ;
- ajouter des oiseaux dans les arbres que votre chat doit attraper ;
- ajouter un prédateur dans les arbres que votre chat doit éviter ;
- de nombreuses améliorations sont possibles, soignez imaginatifs ...

Pour celles et ceux qui voudraient s'entraîner (dans l'objectif de la nuit du code) vous trouverez d'autres environnements de jeu sur studserver.

A vous d'être imaginatifs pour inventer votre propre jeu à partir de ces éléments.



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