Pygame Sprite (Lutin)

Les lutins

Dans pygame un lutin est une image faisant partie d'une scène graphique plus large. Pygame dispose de fonctions spécifiques pour traiter les lutins.

Un lutin est donc un objet sur une scène qui interagit avec d'autres lutins. C'est exactement le concept des lutins de Scratch.

Premier exemple :

Une scène sur laquelle des blocs noirs sont posés. Un bloc rouge guidé par la souris doit collecter les blocs noirs. Le programme comptabilise le score.

Nous allons créer une classe **Bloc** qui **dérive de la classe Sprite de pygame**. La classe Bloc **hérite** de toutes les fonctionnalités de cette classe Sprite.

Le constructeur de notre classe Bloc demande que soient définies la couleur, la largeur et la hauteur du lutin.

```
class Bloc(pygame.sprite.Sprite):
    """cette classe représente un bloc.
        Elle est fille de la classe pygame.sprite.Sprite   """
    #constructeur d'un bloc
    def __init__(self,couleur,largeur,hauteur):
        #appel au constructeur de la classe mère
        super().__init__()
```

Attention : les fonctions init sont encadrées par 2 soulignés consécutifs.

La ligne super().__init__() appelle le constructeur de la classe mère (la classe Sprite), afin que les lutins s'initialisent.

```
        #Création de l'image représentant le lutin.
        #Ici ce sera un rectangle coloré.
        self.image=pygame.Surface([largeur,hauteur])
        self.image.fill(couleur)
```

Ici l'image associée au lutin sera un simple rectangle coloré.

Cette image peut bien évidemment être une image importée.

Dans ce cas on remplacerait ces 2 lignes par :

```
#Création de l'image représentant le lutin.
self.image=pygame.image.load(“ images/joueur.png “).convert_alpha()
```

Dans ce cas, les dimensions du lutin sont mises automatiquement aux dimensions de l'image. Il n'est donc pas nécessaire de passer ces dimensions dans le constructeur.

Cependant nous aurons besoin de connaitre ces dimensions dans la suite du programme.

Pour cela nous pouvons obtenir un objet **Rect**, défini dans pygame, qui représente le rectangle occupé par l'image du lutin.

```
self.rect = self.image.get_rect()
```

Cet objet Rect, dispose d'attributs x et y qui représentent les coordonnées où sera dessiné le lutin.

Si dans mon programme, monLutin est une variable qui référence un lutin du jeu, pour déplacer ce lutin il suffira de donner une valeur à : monLutin.rect.x et monLutin.rect.y.

Début du programme :

```
import pygame
import random
#----------Nos constantes----------
NOIR = (0, 0, 0)
BLANC = (255, 255, 255)
ROUGE = (255, 0, 0)
ECRAN_L = 700
ECRAN_H = 400
#-------La classe Bloc ------------
class Bloc(pygame.sprite.Sprite):
    """cette classe représente un bloc.
        Elle est fille de la classe pygame.sprite.Sprite   """
    #constructeur d'un bloc
    def __init__(self,couleur,largeur,hauteur):
        #appel au constructeur de la classe mère
```

```
        super().__init__()
        #Création de l'image représentant le lutin.
        #Ici ce sera un rectangle coloré.
        self.image=pygame.Surface([largeur,hauteur])
        self.image.fill(couleur)
        self.rect = self.image.get_rect()


#-------Nos variables ----------
arret = False
horloge = pygame.time.Clock()
score = 0
#-------Créer une fenêtre ---------
pygame.init()
ecran = pygame.display.set_mode([ECRAN_L, ECRAN_H])
pygame.display.set_caption(" jeu2 pygame ")
```

Suite du programme

Nous allons travailler avec deux groupes de lutins.

Un premier groupe contiendra tous les blocs noirs du jeu, et le second groupe l'ensemble des lutins du jeu : blocs noirs + joueur.

Pygame permet lorsqu'on a défini un groupe de dessiner, et de déplacer tous les lutins du groupe avec une seule instruction. Il permet également de tester en une instruction, une collision entre un lutin du groupe et un lutin extérieur au groupe.

```
#-------Les listes de lutins ----------
bloc_liste = pygame.sprite.Group()
tout_lutins=pygame.sprite.Group()
```

Créons maintenant tous les blocs noirs et ajoutons-les à bloc_liste :

```
#Création des blocs noirs
for i in range(50):
```

```
    bloc = Bloc(NOIR,20,15)
    #position aléatoire de ce bloc sur la scène
    bloc.rect.x=random.randrange(ECRAN_L)
    bloc.rect.y=random.randrange(ECRAN_H)
    #Ajouter le bloc à nos deux listes de lutins
    bloc_liste.add(bloc)
    tout_lutins.add(bloc)
```

Créons le joueur : il s'agit simplement d'un bloc rouge, que nous ajouterons à la liste tou_lutins.

```
#Création du joueur
joueur = Bloc(ROUGE,20,15)
tout_lutins.add(joueur)
```

La boucle principale

Dans la boucle qui détecte les évènements, nous ne détectons que l'appui sur la fermeture de la fenêtre qui met fin au programme.

```
#----Boucle principale ---------
while not arret:
    #boucle des évènements
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            arret = True
```

Dans notre jeu, le rectangle rouge du joueur doit suivre la souris et collecter des rectangles noirs. Il nous faut donc récupérer les coordonnées de la souris et affecter ces valeurs au lutin représentant le joueur, pour faire bouger ce lutin en même tant que la souris.

Nous allons charger une nouvelle image, que le joueur pourra déplacer avec la souris.

```
    #logique du jeu
    #Récupérer les coordonnées de la souris
    pos = pygame.mouse.get_pos()
```

```
#déplacer le joueur à cette position
joueur.rect.x = pos[0]
joueur.rect.y = pos[1]
```

Il ne reste plus qu'à détecter une collision entre le joueur et un rectangle noir.

La classe sprite propose une fonction qui permet de détecter les collisions entre un lutin et un groupe de lutins présants dans un groupe.

Cette fonction fournit une liste de tous les lutins du groupe entrés en collision. Le dernier paramètre permet d'indiquer si le lutin touché doit être supprimé dans le jeu ou non.

```
#Le joueur est t'il entré en collision avec un bloc noir
collision_liste = pygame.sprite.spritecollide(joueur,bloc_liste,True)
```

La liste collision contient donc la référence à tous les blocs noirs touchés par le joueur.

Il suffit maintenant de tester la longueur de cette liste et d'incrémenter le score en conséquence.

```
#Tester la liste des collisions
    if len(collision_liste)> 0:
        score+=len(collision_liste)
        print(score)
```

Ici le score sera affiché dans la console python, pas dans la fenêtre du jeu. Nous verrons ça plus tard.

Enfin, il faut rafraichir l'écran et dessiner tous les lutins.

Pygam offre une fonction **draw** qui permet d'afficher tous les lutins d'un groupe, en une seule fois. Elle parcourt tous les lutins contenus dans la liste et les affiche.

```
#Code traçant les objets du jeu
 #commencer par effacer l'écran
ecran.fill(BLANC)
#afficher tous les lutins
tout_lutins.draw(ecran)
```

La fin du programme :

```
    #Mettre l'écran à jour
    pygame.display.flip()
    #Mise à jour de l'écran à raison de 60 images/seconde
    horloge.tick(60)

pygame.quit()
```

Comme vous pouvez le constater, pygame offre des fonctions simples permettant de gérer facilement des listes de lutins.

Le programme complet

```
import pygame
import random
#----------Nos constantes----------
NOIR = (0, 0, 0)
BLANC = (255, 255, 255)
ROUGE = (255, 0, 0)
ECRAN_L = 700
ECRAN_H = 400
#-------La classe Bloc ------------
class Bloc(pygame.sprite.Sprite):
    """cette classe représente un bloc.
          Elle est fille de la classe pygame.sprite.Sprite   """
     #constructeur d'un bloc
    def __init__(self,couleur,largeur,hauteur):
        #appel au constructeur de la classe mère
        super().__init__()
        #Création de l'image représentant le lutin.
```

```
        #Ici ce sera un rectangle coloré.
        self.image=pygame.Surface([largeur,hauteur])
        self.image.fill(couleur)
        self.rect = self.image.get_rect()

#-------Nos variables ----------
arret = False
horloge = pygame.time.Clock()
score = 0
#-------Créer une fenêtre ---------
pygame.init()
ecran = pygame.display.set_mode([ECRAN_L, ECRAN_H])
pygame.display.set_caption(" jeu2 pygame ")
#-------Les listes de lutins -----------
bloc_liste = pygame.sprite.Group()
tout_lutins=pygame.sprite.Group()

#Création des blocs noirs
for i in range(50):
    blocn = Bloc(NOIR,20,15)
    #position aléatoire de ce bloc sur la scène
    blocn.rect.x=random.randrange(ECRAN_L)
    blocn.rect.y=random.randrange(ECRAN_H)
    #Ajouter le bloc à nos deux listes de lutins
    bloc_liste.add(blocn)
    tout_lutins.add(blocn)

#Création du joueur
joueur = Bloc(ROUGE,20,15)
```

```
tout_lutins.add(joueur)
#----Boucle principale ---------
while not arret:
    #boucle des évènements
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            arret = True

    #logique du jeu
    #Récupérer les coordonnées de la souris
    pos = pygame.mouse.get_pos()
    #déplacer le joueur à cette position
    joueur.rect.x = pos[0]
    joueur.rect.y = pos[1]
    #Le joueur est t'il entré en collision avec un bloc noir
    collision_liste = pygame.sprite.spritecollide(joueur,bloc_liste,True)
    #Tester la liste des collisions
    if len(collision_liste)> 0:
        score+=len(collision_liste)
        print(score)
    #Code traçant les objets du jeu
   #commencer par effacer l'écran
    ecran.fill(BLANC)
    #afficher tous les lutins
    tout_lutins.draw(ecran)
   #Mettre l'écran à jour
    pygame.display.flip()
    #Mise à jour de l'écran à raison de 60 images/seconde
    horloge.tick(60)
```

```
pygame.quit()
```

Une amélioration du jeu

Nous voulons maintenant que les blocs noirs bougent. A chaque top d'horloge, ils descendent d'un pixel. Si nous ne testons pas si un bloc arrive en bas de l'écran, celui-ci va disparaitre définitivement. Arrivé en bas de l'écran, le bloc doit donc réapparaitre n'importe où sur l'écran.

Nous ajoutons donc une méthode update à la classe Bloc, qui sera appelé dans la boucle principale par **bloc_liste.update()**.

Tous les blocs listés dans bloc_liste seront mis à jour.

La méthode update de la classe Bloc :

```
def update(self):
    # Déplacer le bloc d'un pixel vers le bas
        self.rect.y += 1
        #Si le bloc arrive en bas de l'écran, le remettre n'importe où sur l'écran
        if self.rect.y > ECRAN_H:
            self.rect.y = random.randrange(0, ECRAN_H)
            self.rect.x = random.randrange(0, ECRAN_L)
```

On rajoute dans la boucle principale :

```
    #Code traçant les objets du jeu
    #commencer par effacer l'écran
    ecran.fill(BLANC)
    #mettre à jour tous les blocs noirs
    bloc_liste.update()
    #afficher tous les lutins
    tout_lutins.draw(ecran)
```

Une deuxième amélioration

Lorsqu'ils entrent en collision avec le joueur, nous avons demandé à ce que les blocs noirs disparaissent du jeu. Au départ, nous avons 50 blocs noirs. Au bout de 50 collisions nous n'arrons plus de blocs noirs dans le jeu.

```
collision_liste = pygame.sprite.spritecollide(joueur,bloc_liste,True)
```

Nous allons mettre False à la place de True (les blocs noirs ne seront plus supprimés du jeu), et nous allons modifier la prise en compte des collisions.

Lorsqu'un bloc noir a été touché, il doit réinitialiser sa position.

```
#Le joueur est t'il entré en collision avec un bloc noir
collision_liste = pygame.sprite.spritecollide(joueur,bloc_liste,False)
#Tester la liste des collisions
for blocn in collision_liste:
    score+=1
    print(score)
    #réinitialiser les positions des blocs touchés
    blocn.reset()
```

Et il nous faut rajouter une nouvelle méthode **reset()** à la classe Bloc :

```
def reset(self):
    self.rect.y = random.randrange(0, ECRAN_H)
    self.rect.x = random.randrange(0, ECRAN_L)
```