

PROGRAMMATION D'UN JEU VIDEO

Objectif : Développer un jeu vidéo simple, **le concept devant être en rapport avec les bateaux.**

Travail demandé :

- Suivre le tutoriel ci-après pour vous familiariser avec le langage Python et Pygame,
- Proposer un concept de jeu que vous développerez d'ici à la fin du semestre,
- Finaliser et présenter le jeu lors de la semaine banalisée.

Modalités :

- Le travail se fera par groupe, en reprenant ceux du jeu d'entreprise,
- Aucun rapport ne sera demandé concernant la phase de développement,
- Le code sera rendu directement dans le **casier électronique** de E. Rodes.

En cas de problème insurmontable : emmanuel.rodes@unilim.fr

1 Introduction

Le but de ce tutoriel est de vous familiariser suffisamment avec le langage Python et plus particulièrement avec la librairie Pygame, pour que vous soyez capables d'écrire un jeu vidéo simple. Le but n'est pas de faire de vous des programmeurs Python, mais plutôt de vous faire découvrir ce langage de manière ludique et informelle.

De par la brièveté de ce document, les informations que vous y trouverez seront nécessairement fragmentaires et certains concepts seront survolés. Vous avez tout intérêt à vous référer soit à des ouvrages de bibliothèque, soit à des sites Internet pour compléter les notions que vous trouverez ici. Pour ce qui est de la bibliographie numérique, vous pouvez notamment vous référer aux sites officiels suivants :

pour Python : www.python.org

pour Pygame : www.pygame.org

Le hors-série de Linux Pratique de février/mars 2012 propose également une très bonne introduction au langage Python en général, avec un dossier consacré à Pygame.

1.1 Le langage Python

Le langage python est un langage relativement récent, puisque la première version date de 1989, et la première version en licence GPL de 2000. Actuellement 2 versions cohabitent, la 2.7 et la 3.3, dans le cadre de ce projet, vous devrez utiliser la version 2.7. Les caractéristiques principales de ce langage en vue de l'utilisation que vous en aurez sont les suivantes :

- **langage interprété** : comme tout langage interprété, Python ne nécessite pas de compilation, mais par contre **requiert l'installation d'un interpréteur** (cf le site officiel). De ce fait, il est également multiplate-forme, et peut être utilisé sous Windows, Linux, Mac. Vos programmes auront une extension .py et devront être lancés via l'interpréteur.
- **typage dynamique** : ce terme signifie que contrairement au C, Python ne nécessite pas la déclaration du type d'une variable avant son utilisation. Le type d'une variable est directement déduit de sa première utilisation, mais ne peut plus changer par la suite.
- **syntaxe** : la syntaxe du Python est relativement proche du C, l'accent ayant été mis sur la simplicité et la lisibilité du code. Vous ne devriez donc pas rencontrer de problème pour vous mettre à ce langage.
- **indentation** : dans un souci de lisibilité et de compacité du code, les accolades utilisées en C ont été supprimées, et remplacées par l'indentation. L'écriture, par exemple, d'un **if** en Python va donc ressembler à cela :

```
if(test):  
    instructions1  
else:  
    instructions2
```

Les “:” remplacent l'accolade ouvrante, et la fin de l'indentation remplace l'accolade fermante. Cela signifie que si votre code n'est pas indenté, non seulement il sera illisible, mais en plus il ne fonctionnera pas du tout car l'interpréteur ne pourra pas le comprendre.


1.2 La bibliothèque Pygame

La bibliothèque Pygame est une des multiples extensions de Python, et comme son nom l'indique, elle est consacrée exclusivement à la création de jeu vidéo. Lister les types de jeu pouvant (et étant) développés à l'aide de Pygame est tout simplement impossible, vous pourrez le constater en regardant les exemples présentés sur le site officiel.

Sur ce dernier (en anglais comme il se doit), vous trouverez une abondante documentation, ainsi que la liste de toutes les fonctions proposées par Pygame, réparties par fonctionnalité. Dans le cadre de

ce projet, le présent tutoriel vous donnera un petit aperçu de l'utilisation de cette librairie et vous permettra par la suite de vous référer à la documentation officielle plus facilement.

2 Les animations avec Pygame

Ici commence le tutoriel proprement dit. Son but n'est pas de vous faire développer un jeu vidéo, mais de vous faire découvrir les fonctions qui vous permettront de le faire. Les quelques images utilisées pour ce tutoriel sont disponibles dans la zone publique de l'iut, mais vous pouvez bien sûr les remplacer par celles que vous voulez. Les différentes portions de code présentées ici **ne sont pas à taper telles quelles**, le symbole  indique les moments où vous devez tester votre programme.

2.1 Un affichage simple

Pour commencer ce tutoriel, ouvrez un nouveau document .py avec l'éditeur de texte de votre choix et sauveez-le dans un dossier où vous aurez placé les images récupérées sur la zone publique. Pour l'utilisation sous Linux, il suffira alors d'appeler python depuis une console sur votre fichier. Pour Windows, voir le site <http://guigui.developpez.com/Tutoriel/Python/PythonWindows/>. Pensez à utiliser un nouveau fichier pour chaque partie, de manière à garder les différentes versions du code, certaines portions sont réutilisées durant le tutoriel.

Mise en place de la fenêtre

De la même manière que pour un programme en C, ou dans tout autre langage, il faut commencer par définir les bibliothèques que l'on souhaite utiliser. Dans le cas présent, on va charger les bibliothèques sys et pygame, cette dernière devant de plus être initialisé pour pouvoir être utilisée :

```
import sys,pygame
from pygame.locals import *
pygame.init()
```

Il faut ensuite définir la taille de la fenêtre qui sera affichée, via sa largeur et sa longueur, toutes deux définies en pixel, le pixel dans le coin supérieur gauche étant le pixel (0,0) :

```
largeur=640
hauteur=480
ecran=pygame.display.set_mode((largeur,hauteur))
```

Afin de pouvoir faire évoluer simplement notre programme par la suite, on va directement

effectuer un affichage dynamique, c'est à dire que notre fenêtre sera redessinée à intervalle régulier, plusieurs fois par seconde. De cette manière il sera très facile de créer des animations à partir de ce premier programme. Pour ce faire, on va commencer par définir une horloge qui permettra de mesurer le temps :

```
clock=pygame.time.Clock()
```

La partie principale de notre programme sera alors simplement une boucle infinie d'affichage qui contiendra les instructions nécessaires au dessin de la fenêtre. Cette boucle sera exécutée en permanence, tant que le programme restera ouvert :

```
while 1:
    clock.tick(60)

    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            sys.exit()

    #Ici viendra le code définissant l'affichage proprement dit

    pygame.display.flip()
```

- La première ligne sert à ralentir de rythme d'exécution de la boucle d'affichage à 60 fois par seconde, afin de limiter la vitesse de l'affichage.
- La boucle **for** va parcourir l'ensemble des événements survenus depuis la dernière boucle (mouvement de la souris, l'appui sur une touche, clic, etc.) et agir en conséquence. Pour l'instant ceci ne sera utilisé que si l'événement est la fermeture de la fenêtre (événement `pygame.QUIT`), auquel cas le programme va s'arrêter (`sys.exit()`).
- La dernière ligne demande l'actualisation de l'affichage. Toutes les modifications de l'affichage se font en fait dans un premier temps dans un buffer d'affichage qui est basculé vers l'écran par cette commande.

Création d'un affichage

Pour l'instant, rien n'est affiché dans notre fenêtre, puisqu'il n'y a rien dans la boucle **while**. Dans un premier temps, on va se contenter de créer un simple fond coloré uni. On va donc tout d'abord définir la couleur que l'on souhaite utiliser pour le fond, hors de la boucle. Au niveau de la boucle d'affichage, on demandera alors à ce que cette couleur soit utilisée pour remplir la fenêtre :

Hors de la boucle:

```
couleur=(10,29,96)
```

Dans la boucle:

```
ecran.fill(couleur)
```

Le code complet de ce premier exemple ressemble donc à ceci, une fois toutes les lignes mises ensembles :

```
import sys,pygame
pygame.init()

largeur=640
hauteur=480
ecran=pygame.display.set_mode((largeur,hauteur))

couleur=(10,29,96)

clock=pygame.time.Clock()

while 1:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            sys.exit()

    écran.fill(couleur)

    pygame.display.flip()
```



Saisissez ce code et vérifiez que vous obtenez bien l'affichage d'une fenêtre bleu.

2.2 Ajout d'éléments

On va maintenant chercher à afficher une image sur notre fond bleu. Pour afficher une image,

pygame a besoin de 2 éléments, l'image elle-même (le motif qui va être affiché), et un objet de type *rect* qui correspond au cadre de l'image, c'est à dire à sa taille et à sa position. Ces 2 éléments peuvent simplement être obtenus via les lignes suivantes :

Hors de la boucle :

```
image=pygame.image.load("ball.png")
rect=image.get_rect()
```

Comme vous pouvez le voir, l'objet de type *rect* est créé simplement par l'appel à la fonction `get_rect()`. Pour afficher simplement l'image, il suffit alors d'inclure l'instruction correspondante dans la boucle d'affichage :

Dans la boucle d'affichage :

```
ecran.blit(image,rect)
```

Cette ligne demande l'affichage de l'image ***image*** à l'emplacement ***rect***. Comme nous n'avons pas déplacé l'objet ***rect***, l'image sera par défaut positionnée en (0,0), c'est à dire dans le coin supérieur gauche de la fenêtre.



Saisissez ce code et vérifiez que la balle s'affiche correctement.

On va maintenant chercher à afficher non plus une image, mais un texte dans notre fenêtre. Le principe va être similaire à celui utilisé pour l'affichage d'une image, à savoir d'un coté le texte, et de l'autre sa position à l'écran définie via un objet de type *rect* :

```
font=pygame.font.Font(None,36)
text=font.render("Bonjour",1,(1,1,1))
textRect=text.get_rect()
...
ecran.blit(text,textRect)
```

- La première ligne sert définir la police de caractère qui va être utilisée, *None* signifiant la police système par défaut et 36 étant la taille désirée,
- La seconde ligne crée l'affichage correspondant au texte choisi, les autres paramètres étant l'antialiasing (0 ou 1) et la couleur du texte,
- Les deux autres lignes sont complètement identiques à celles vues précédemment pour les images.



Saisissez ce code et vérifiez que vous obtenez bien l'affichage attendu.

2.3 Animation

Le principe d'une animation est relativement simple, on affiche une image à un endroit donné, puis on l'efface avant de la ré-afficher à côté de l'emplacement initial. On répète alors simplement ces étapes d'affichage/effacement suffisamment vite pour que l'œil humain interprète cela comme un déplacement continu.

Pour réaliser une animation à partir du programme précédent, affichant la balle, il suffit donc de rajouter dans la boucle d'affichage les lignes permettant effacement de l'image et son déplacement. En fait, dans notre cas, l'effacement de l'image se fait déjà lorsque l'on repeint la fenêtre avec la couleur d'arrière-plan. Pour obtenir une animation, il faut donc uniquement placer le code suivant dans la boucle d'affichage :

```
ecran.fill(couleur)
rect=rect.move(1,0)
ecran.blit(image,rect)
```



Modifiez votre code et vérifiez que la balle se déplace.

Comme vous avez pu le voir, la balle est déplacée d'un pixel à chaque itération de la boucle *while*, c'est à dire qu'elle est déplacée à la vitesse de 60 pixels/s vers la droite de l'écran. Le résultat est une animation très lente et une balle qui sort de l'écran. Afin d'obtenir un résultat un peu plus satisfaisant, on va utiliser le code suivant :

Hors de la boucle:

```
vitesse=[5,5]
```

Dans la boucle:

```
ecran.fill(couleur)
```

```

if((rect.left<0) or (rect.right>largeur)):
    vitesse[0]=-vitesse[0]
if((rect.top<0) or (rect.bottom>hauteur)):
    vitesse[1]=-vitesse[1]
rect=rect.move(vitesse)

ecran.blit(image,rect)

```

- On définit une variable **vitesse** qui contient la vitesse horizontale et la vitesse verticale de la balle, ces 2 valeurs valant 5, ce qui correspond à 300 pixels/s,
- Dans la cas où la balle va sortir de la fenêtre par la gauche (`rect.left<0`), la droite (`rect.right>largeur`), le haut (`rect.top<0`) ou le bas (`rect.bottom>hauteur`), on inverse simplement le sens de déplacement du ballon : `vitesse=-vitesse`.



Saisissez ce code et vérifiez que votre balle se déplace rapidement en rebondissant au sein de la fenêtre, sans en sortir.

2.4 Animations multiples

Le code écrit dans la partie précédente fonctionne parfaitement et reste relativement simple. Si l'on désire par contre animer plusieurs balles simultanément, il va falloir déclarer autant de variables **vitesse** que de balles, et répéter également les lignes de l'encadré précédent. Pour pouvoir conserver un code simple et lisible, on va réécrire la version précédente du programme en utilisant les concepts objets.

Le principe est de définir un modèle de balle (on dit une *classe*) qui possédera un certain nombre d'*attributs* et un certain nombre de *méthodes* (c'est à dire des fonctions qui modifieront son état). Pour chaque balle que l'on voudra animer, il suffira de créer un nouvel exemplaire (on dit une instance) à partir du modèle, et cet exemplaire pourra être affiché directement.

Pour pouvoir réaliser l'animation précédente, nous avons eu besoin d'utiliser les variables **image**, **rect** et **vitesse** qui correspondaient respectivement à l'image à dessiner, à sa position et à sa vitesse de déplacement. Il a également fallut définir le mouvement de la balle en fonction de sa vitesse et des limites de la fenêtre. Notre classe **Balle** va donc posséder des attributs **image**, **rect** et **vitesse**, ainsi qu'une méthode **deplacement** :

Code à placer au début du programme:

```

class Balle():

    def __init__(self,argumentVitesse):

```



```

        self.vitesse=argumentVitesse
        self.image=pygame.image.load("ball.png")
        self.rect=self.image.get_rect()

    def deplacement(self):
        if((self.rect.left<0) or (self.rect.right>640)):
            self.speed[0]=-self.speed[0]
        if((self.rect.top<0) or (self.rect.bottom>480)):
            self.speed[1]=-self.speed[1]

        self.rect=self.rect.move(self.speed)

```

La classe **Balle** ne fait donc que reprendre les lignes déjà écrites, on peut simplement noter que la valeur de l'attribut **vitesse** est fixée par le passage du paramètre **argumentVitesse**. Ce dernier devra être donné lors de chaque création d'une instance de la classe **Balle**. Pour utiliser notre toute nouvelle classe afin de reproduire l'animation précédente, il suffit alors d'utiliser le code suivant :

```

speed=[5,5]
ecran=pygame.display.set_mode((640,480))
couleur=(10,29,96)

balle1=Balle(speed) #creation d'une instance
clock=pygame.time.Clock()

while 1:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            sys.exit()

    écran.fill(couleur)
    balle1.deplacement() #appel de la méthode deplacement
    écran.blit(balle1.image,balle1.rect)
    pygame.display.flip()

```

Ce code permet de créer une instance de la classe **Balle** nommée **balle1** et qui aura une vitesse de déplacement de [5,5]. A chaque itération, on appellera la méthode **deplacement** de **balle1** qui gère son propre déplacement, avant de retracer l'image. On obtient donc un résultat complètement similaire à celui du code précédent, mais en utilisation des concepts objets.



Saisissez ce code et vérifiez que vous obtenez bien l'affichage précédent. Modifiez-le ensuite, de manière à avoir 2 balles à l'écran, avec des déplacements différents.

2.5 Groupes d'animation

L'utilisation de concepts objets a déjà permis de simplifier grandement l'écriture de l'animation à 2 balles. Si l'on souhaite animer un grand nombre d'objets, même en utilisant les classes cela va se révéler rapidement fastidieux. Pygame permet de travailler directement avec des groupement d'objets, et notamment d'afficher tous les objets d'un groupe avec une seule fonction.

Pour cela, il est tout d'abord nécessaire de modifier quelque peu notre classe **Balle**, sans que cela ne change rien au fonctionnement actuel du programme :

```
class Balle(pygame.sprite.Sprite):  
  
    def __init__(self, vitesse):  
        pygame.sprite.Sprite.__init__(self)  
        self.speed=vitesse  
        self.image=pygame.image.load("ball.gif")  
        self.rect=self.image.get_rect()  
  
    def update(self):  
        if((self.rect.left<0) or (self.rect.right>640)):  
            self.speed[0]=-self.speed[0]  
        if((self.rect.top<0) or (self.rect.bottom>480)):  
            self.speed[1]=-self.speed[1]  
  
        self.rect=self.rect.move(self.speed)
```



Faites la modification suggérée et vérifiez que l'animation fonctionne toujours.

Grâce à cette modification, on va pouvoir créer un groupement d'objets, qui sera une classe particulière de type **RenderPlain**. On va par exemple essayer d'afficher non plus 2 mais 4 balles à l'écran, via l'utilisation d'un groupement :

Hors de la boucle :

```
groupBalles=pygame.sprite.RenderPlain()  
groupBalles.add(Balle([1,1])  
groupBalles.add(Balle([0,2])  
groupBalles.add(Balle([3,1])
```

```
groupBalles.add(Balle([3,0]))
```

- on crée un objet **groupBalles** qui est de type `RenderPlain`,
- on peut alors simplement ajouter 4 instances de **Balle** à ce groupe, chaque balle ayant sa vitesse de déplacement propre.

Pour tout ce qui est affichage, le fait d'utiliser un groupe va simplifier le code de manière remarquable, puisqu'il ne va rester que les lignes suivantes :

Dans la boucle d'affichage:

```
ecran.fill(couleur)
groupBalles.update()
groupBalles.draw(ecran)
pygame.display.flip()
```

- la ligne `groupBalles.update()` appelle la fonction `update()` de tous les balles du groupe,
- la ligne `groupBalles.draw()` appelle la fonction `blit()` sur tous les balles du groupe,
- au final, on gère l'affichage d'autant de balles qu'on le souhaite en 2 lignes de code.



Modifiez votre programme pour afficher autant de balles que vous le souhaitez.

3 Un jeu avec Pygame

3.1 Interface utilisateur : le clavier

Pour créer un jeu, il est nécessaire que l'utilisateur puisse interagir avec l'affichage, ce qui se fait classiquement soit via le clavier, soit via la souris. On va maintenant reprendre le code précédent afin qu'il affiche une balle (ce que l'on sait déjà faire), en faisant en sorte que ses mouvements répondent à l'appui sur les flèches du clavier. Le code de départ sera donc le suivant :

```
class Balle(pygame.sprite.Sprite):
    def __init__(self, speed):
        pygame.sprite.Sprite.__init__(self)
        self.vitesse=speed
        self.image=pygame.image.load("ball.gif")
        self.rect=self.image.get_rect()
```

```

def update(self):
    if((self.rect.left<0) or (self.rect.right>640)):
        vitesse[0]=-vitesse[0]
    if((self.rect.top<0) or (self.rect.bottom>480)):
        vitesse[1]=-vitesse[1]
    self.rect=self.rect.move(self.vitesse)

largeur=640
hauteur=480
speed=[5,5]
ecran=pygame.display.set_mode((largeur,hauteur))
couleur=(10,29,96)
balle=Balle(speed)
clock=pygame.time.Clock()

while 1:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            sys.exit()

    ecran.fill(bleu)
    balle.update()
    ecran.blit(balle.image,balle.rect)
    pygame.display.flip()

```

Toutes les actions utilisateurs sont gérées directement par Pygame, qui les renvoie dans la file d'attente des événements. Pour l'instant, on ne testait que l'action de fermeture de la fenêtre, c'est pourquoi les frappes claviers étaient pour l'instant ignorées par notre programme :

```

if event.type==pygame.QUIT:
    sys.exit()

```

Pour que les appuis sur les flèches du clavier soit pris en compte, il suffit donc de les rajouter dans les événements testés. On va maintenant faire en sorte que le déplacement du ballon soit modifié, ce qui se fait simplement ainsi :

```

if event.type==pygame.QUIT:
    sys.exit()
elif event.type== KEYDOWN:
    if event.key==K_LEFT:
        balle.vitesse[0]-=1

```

```
elif event.key==K_RIGHT:
    balle.vitesse[0]+=1
elif event.key==K_UP:
    balle.vitesse[1]-=1
elif event.key==K_DOWN:
    balle.vitesse[1]+=1
```

- Dans le cas où on appuie sur une touche, le programme va tester si la touche en question est une des flèches (K_LEFT, K_RIGHT, K_UP ou K_DOWN),
- Selon la touche enfoncée, la vitesse du ballon (*balle.vitesse*) sera modifiée en conséquence.



Modifiez votre programme et vérifiez son fonctionnement.

Vous avez pu constater qu'un appui long sur une touche sera traité comme un appui unique. Il est donc nécessaire de ré-appuyer à chaque fois. Ceci pouvant être gênant dans le cadre d'un jeu, il est possible de l'éviter en incluant simplement la ligne suivante :

Hors de la boucle:

```
pygame.key.set_repeat(10, 10)
```

- le premier nombre correspond au délai nécessaire (en ms) avant que Pygame ne considère un appui long comme 2 appuis sur la touche,
- le second nombre correspond au délai entre les “appuis” suivants,
- ces chiffres peuvent être adaptés pour avoir un comportement plus ou moins “réactif” du jeu.

3.2 Interface utilisateur : la souris

Changer l'icône de la souris

L'interaction avec la souris va être nécessaire pour la plupart des jeux, soit déplacer un objet avec la souris, soit pour qu'un objet puisse être cliqué. Le premier type d'interaction est très simple à implémenter, il suffit de retracer systématiquement l'objet souhaité à la nouvelle position de la souris. Pour obtenir ce résultat, il suffit d'adapter quelque peu notre classe *Balle* pour créer une nouvelle classe baptisée (par exemple) *Target* :

```
class Target(pygame.sprite.Sprite):

    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image=pygame.image.load("target.png")
        self.rect=self.image.get_rect()

    def update(self):
        pos=pygame.mouse.get_pos()
        self.rect.center=pos
```

- la fonction `pygame.mouse.get_pos()` permet de récupérer la position de la souris,
- il suffit alors de placer le centre de notre objet à cette position.

Ce code est tout à fait fonctionnel, par contre on va obtenir une icône de cible superposée à l'icône de la souris, ce qui n'est pas satisfaisant visuellement parlant. On a donc intérêt à masquer l'icône "normale" de la souris, ce qui se fait simplement ainsi:

Hors de la boucle d'affichage:

```
pygame.mouse.set_visible(0)
```



Écrivez le nouveau programme et vérifiez que l'icône de la souris a changée.

Cliquer sur un objet

Traiter un clic de souris sur un objet est à peine plus compliqué avec Pygame, puisque le cas échéant un événement de type `MOUSEBUTTONDOWN` va être placé automatiquement placé dans la file des événements. On va par exemple afficher à nouveau une balle, et faire en sorte qu'en cliquant dessus on puisse la faire disparaître. On va repartir du code suivant, en incluant bien sur la définition des classes **Balle** et **Target** qui n'apparaissent pas ici :

```
largeur=640
hauteur=480
speed=[5,5]
ecran=pygame.display.set_mode((largeur,hauteur))
couleur=(10,29,96)
pygame.mouse.set_visible(0)
```

```

target=Target()
balle=Balle(speed)
groupBalles=pygame.sprite.RenderPlain()
groupBalles.add(balle)
clock=pygame.time.Clock()

while 1:
    clock.tick(60)
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            sys.exit()

    ecran.fill(couleur)
    target.update()
    groupBalles.update()
    groupBalles.draw(ecran)
    ecran.blit(target.image, target.rect)
    pygame.display.flip()

```

Ce code affiche simplement une balle qui rebondit à l'intérieur de la fenêtre, avec une icône de souris modifiée. Bien que l'on affiche qu'une seule balle, ce programme utilise un groupement appelé **groupBalles** de type `RenderPlain`.

Pour faire en sorte qu'un clic sur la balle la fasse disparaître, il suffit de rajouter les lignes correspondantes dans le traitement des événements :

```

if event.type==MOUSEBUTTONDOWN:
    pos=pygame.mouse.get_pos()
    if((pos[0]>balle.rect.left) and (pos[0]<balle.rect.right) and
        (pos[1]>balle.rect.top) and (pos[1]<balle.rect.bottom)):
        balle.kill()

```

- dans le cas d'un clic de la souris, on vérifie que l'on a cliqué sur le ballon,
- si c'est le cas, on appelle la fonction **kill()**, qui supprime **balle** du groupement **groupBalles**,
- comme seul **groupBalles** est affiché, le ballon disparaît...



Écrivez le nouveau programme et vérifiez son fonctionnement.

3.3 Interactions entre les objets

Le traitement de l'interaction entre la souris et la balle de l'exemple précédent est relativement lourd, et en tout état de cause, complètement inutilisable si on veut tester l'interaction entre de multiples objets.

Pour gérer ce genre d'interaction, le plus simple est d'utiliser les fonctions proposées par Pygame. On va par exemple afficher à nouveau un grand nombre de balles, et essayer de les « attraper ». Le code de départ va être celui affichant 4 balles, que vous modifierez pour commencer avec 6 balles. On va également ajouter une classe **Robot** (identique à la classe **Target**) pour le personnage:

```
class Robot(pygame.sprite.Sprite):  
  
    def __init__(self):  
        pygame.sprite.Sprite.__init__(self)  
        self.image=pygame.image.load("robot.png")  
        self.rect=self.image.get_rect()  
  
    def update(self):  
        pos=pygame.mouse.get_pos()  
        self.rect.center=pos
```

A chaque tour de la boucle d'affichage, on va alors devoir tester si notre robot est en contact avec une des balles et, le cas échéant, faire disparaître la balle qui a été « attrapée » :

```
pygame.sprite.spritecollide(robot,groupBalles,True)
```

- la fonction **spritecollide** vérifie si **robot** est en contact avec les éléments de **groupBalles**,
- le cas échéant, la fonction **kill()** est appelé automatiquement (**True**) sur la balle en contact avec le robot,
- la fonction **spritecollide** teste le chevauchement entre les objets *rect* et pas les images, il faut en tenir compte pour un résultat optimal.



Écrivez ce dernier programme en faisant en sorte que les balles apparaissent en bas à droite de l'écran, au lieu de en haut à gauche pour pouvoir en tester le fonctionnement.

Vous êtes maintenant au bout de ce tutoriel et vous connaissez suffisamment de bases de Python et de Pygame pour commencer à réfléchir à votre projet de jeu vidéo. **Vous ne pourrez pas** vous contenter de ces bases pour développer un jeu quel qu'il soit, mais vous devez maintenant être capables de suivre les tutoriels du site officiel de Pygame pour y piocher les informations dont vous aurez besoin pour la suite du projet.