Programmer une animation avec

Pygame est une librairie Python écrite par Pete Shinners au début des années 2000 qui s'appuie sur la librairie C (mais aussi disponible dans d'autres langages que C) **SDL** (**S**imple **D**irectmedia **L**ibrarie).

Elle permet de gérer l'affichage d'images dans une fenêtre, le temps, le clavier, la souris, les joysticks, et le son, le tout en théorie de façon totalement portable entre les différents OS.

C'est sans doute un bon choix pour s'exercer en Python, mais pas forcément le meilleur si on veut uniquement faire de la programmation créative et que Python n'est pas un objectif en soit
(voir par exemple Processing : https://processing.org).

Ressources sur Pygame en français

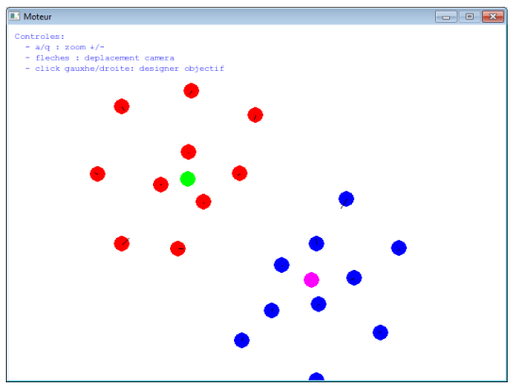
Apprendre la programmation par le jeu
Vincent Maille. Ellipse.

Concevez des jeux avec Pygame (developpez.com)
de Alexandre Galode, traduction française du livre Make Games with Python de Sean M. Tracey
https://deusyss.developpez.com/tutoriels/Python/Pygame/Ch01/

Interface graphique Pygame pour Python (openclassrooms)
https://openclassrooms.com/courses/interface-graphique-pygame-pour-python
voir aussi les discussions sur Pygame sur les forums d'openclassroom

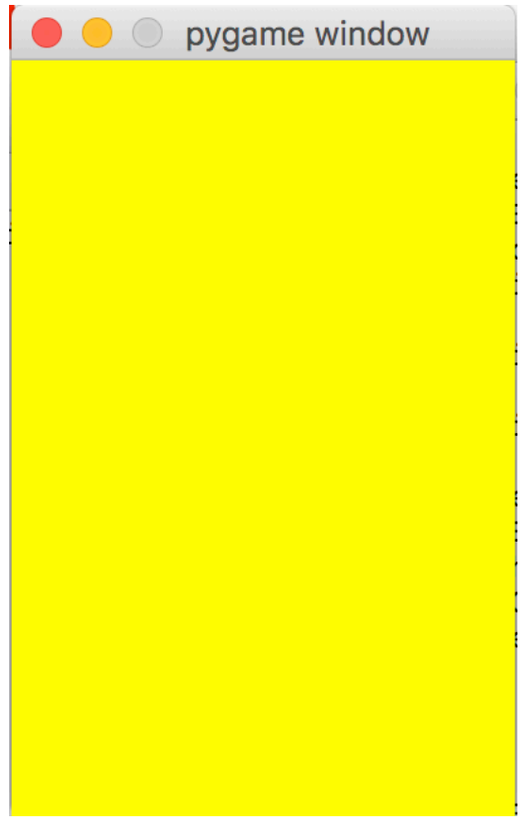
Atelier formation ISN de Vincent Thomas (INRIA Nancy)
https://members.loria.fr/VThomas/mediation/ISN_moteur_2017/

intéressant en particulier pour la simulation de systèmes multi-agents



Premier programme Pygame

```
# importation du module et des constantes locales
import pygame
from pygame.locals import *

# initialisation (à ne pas oublier!)
pygame.init()

# creation de la surface "fenêtre"
dimensions = (largeur, hauteur) = (200, 300)
fenetre = pygame.display.set_mode(dimensions)

# dessin dans la surface fenêtre
jaune = (255, 255, 0) # le code RGB de la couleur jaune
fenetre.fill(jaune)

# affichage
pygame.display.update()
pygame.event.get()  # pas necessaire, en theorie...
pygame.time.wait(5000) # attend 5000 ms avant de terminer
pygame.quit()
```



Ce programme affichera une fenêtre jaune pendant 5 secondes

Les Couleurs

Pour définir une couleur, on utilise la « fonction » Color(...). Cette fonction peut prendre un triplet (r,g,b) représentant le code RGB de la couleur; le jaune s'obtient comme mélange de rouge et de vert.

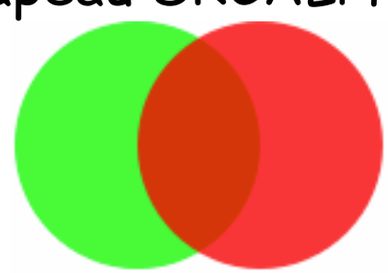
```
jaune = Color(255, 255, 0) # le code RGB de la couleur jaune
```

Color peut aussi prendre une **chaîne de caractères** :

```
gris = Color("gray") # la couleur grise
```

"red" "blue" "black" "green" "white" "gray" "yellow" *etc*

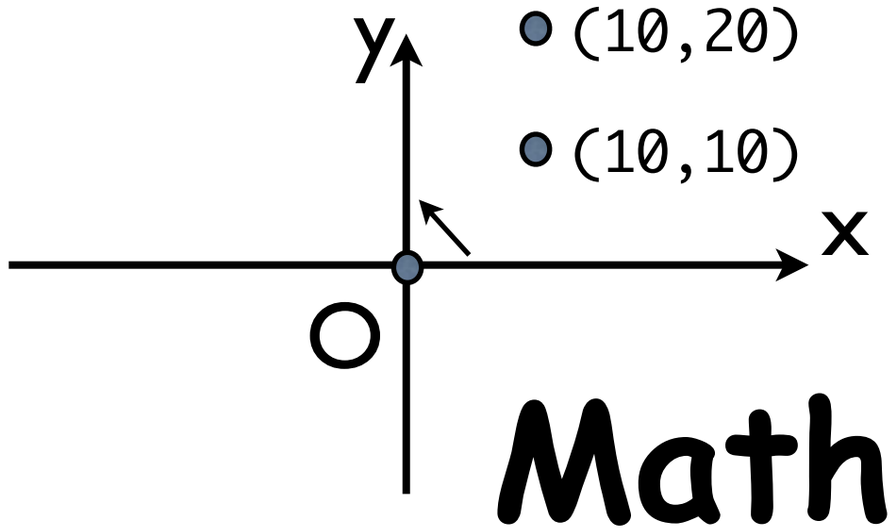
Color peut aussi prendre un quadruplet RGBA: le quatrième argument représente la **transparence**. Il doit être choisi entre 0 (transparent) et 255 (opaque).
Important: il faudra alors travailler avec des surfaces créées avec le drapeau SRCALPHA pour avoir une transparence à l'échelle du pixel (voir doc).

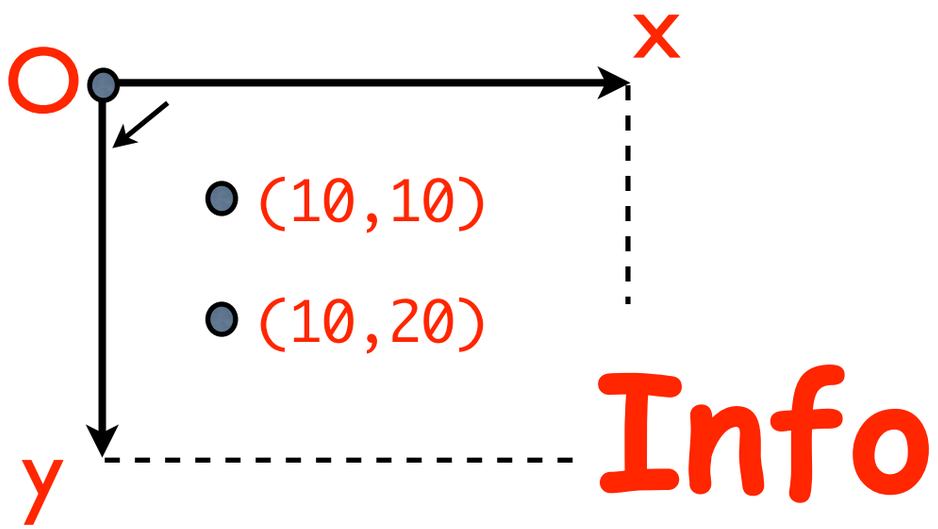


```
vert_transparent = Color(0,255,0,128) # un vert transparent
```

Coordonnées

- ATTENTION : la plupart des langages de programmation graphiques n'utilisent pas les axes mathématiques usuels !





Pygame ne déroge pas à la règle. On travaille en coordonnées informatiques

Dessiner un rectangle

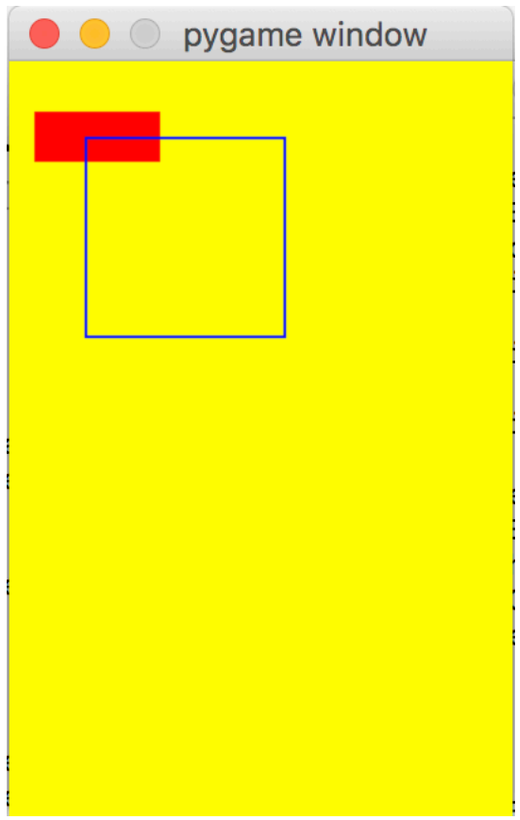
On commence par déclarer un objet Rect

```
rect1 = Rect(10,20,50,20)
```

coordonnées angle supérieur gauche — largeur — hauteur

Puis on appelle la fonction rect du sous-module draw pour le dessiner sur une surface (ici la fenêtre) en précisant la couleur de remplissage

```
pygame.draw.rect(fenetre, rouge, rect1)
```



On peut omettre l'objet rect et le remplacer par un tuple. On peut aussi préciser une largeur de trait, ce qui annule le remplissage.

```
pygame.draw.rect(fenetre, bleu, [30,30,80,80], 1)
```

Dessiner un cercle ou une ellipse

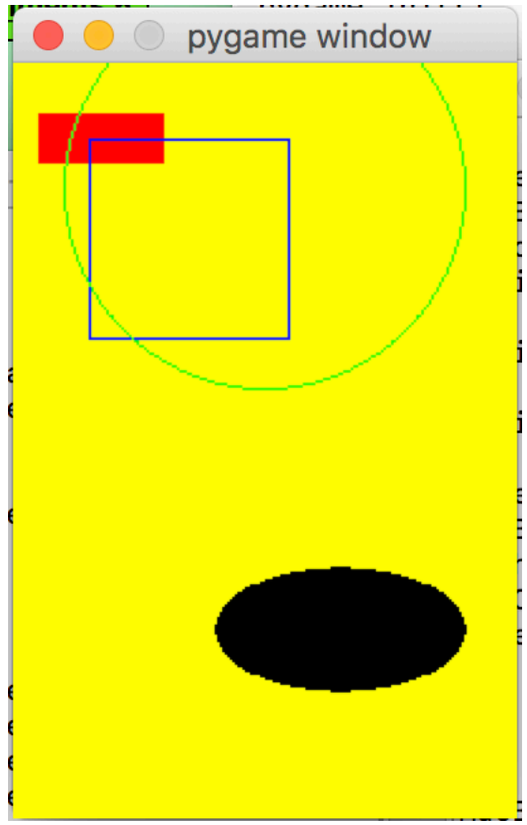
```
pygame.draw.circle(fenetre, vert, (100,150), 80, 1)
```

coordonnées du centre rayon épaisseur

De même que pour le rectangle, l'épaisseur est un paramètre optionnel, par défaut égal à 0, i.e. remplissage

Pour dessiner une ellipse, on donne le rectangle qui la contient.

```
rect2 = Rect(80, 200, 100, 50)
pygame.draw.ellipse(fenetre, noir, rect2)
```



Dessiner un segment ou une ligne brisée

```
pygame.draw.line(fenetre, rouge, (100,150), (80,200) )
```

départ

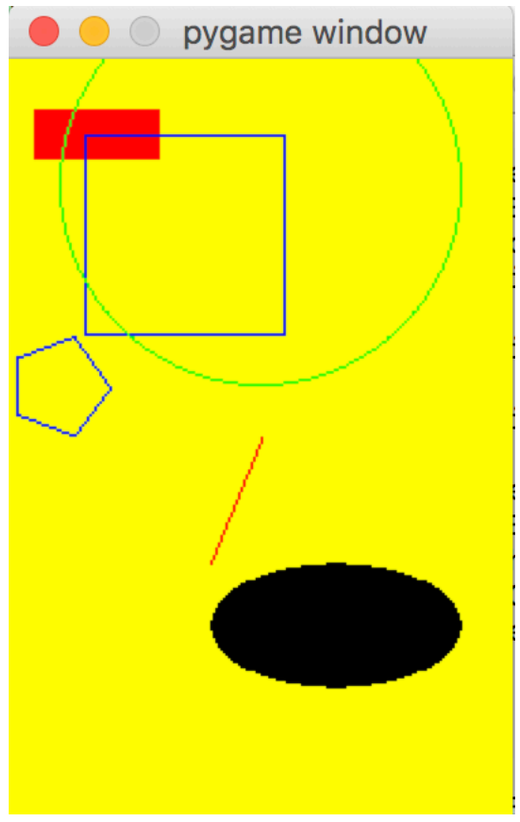
arrivée

Pour dessiner une ligne brisée, on utilise pygame.draw.lines.

```
lines(Surface, color, closed, pointlist, width=1)
```

- pointlist est une liste de points
- si closed est vrai, le premier et le dernier sont reliés

```
L1 = [(cos(2*i*pi/5), sin(2*i*pi/5)) for i in range(5)]
L2 = [(20*x+20,20*y+130) for (x,y) in L1]
L3 = [(int(x), int(y)) for (x,y) in L2]
pygame.draw.lines(fenetre, bleu, True, L3)
```

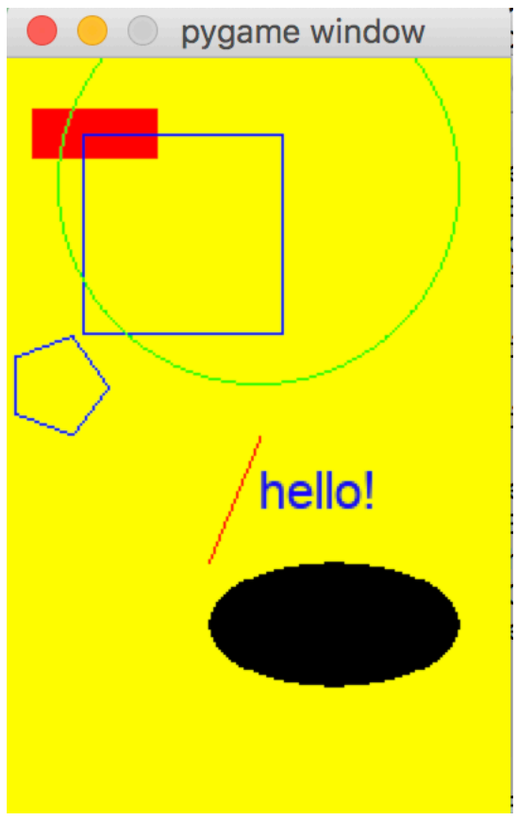


Ajouter un texte

Pour pouvoir écrire un texte, il faut tout d'abord sélectionner une fonte.
Deux méthodes sont possibles:

- on fait confiance à Pygame pour trouver la fonte dans l'installation locale
- on spécifie le chemin vers le fichier de la fonte qui nous intéresse

```
arial1 = pygame.SysFont("arial", 20)
arial2 = pygame.Font("/Library/Fonts/Arial.ttf",20)
```



On peut ensuite utiliser la fonte pour construire une nouvelle surface qui contient un texte à l'aide de la méthode **render**.

```
texte = arial1.render("hello!", True, bleu)
```

Un quatrième paramètre optionnel permet de fixer une couleur de fond.

antialising

couleur

On colle enfin ce texte dans la fenêtre à l'aide de **blit**

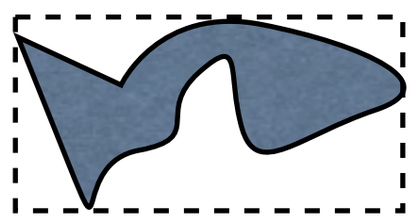
```
fenetre.blit(texte, (100,160) ) # position du coin supérieur gauche
```

Collage avec blit, notion de bounding box

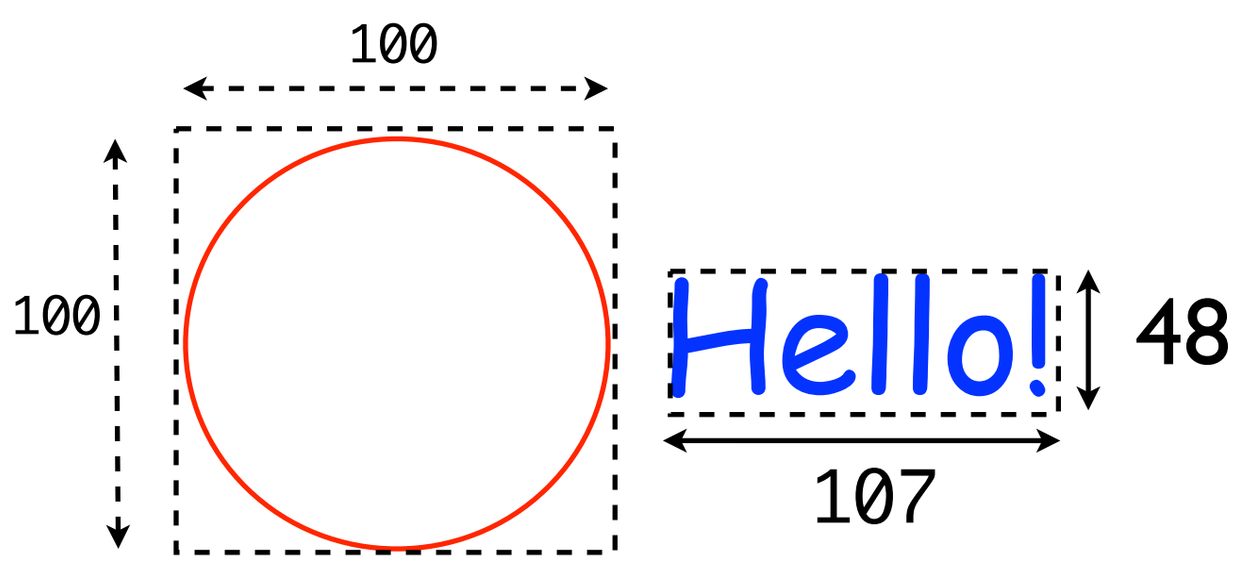
La méthode blit permet de coller une surface dans une autre.

```
fenetre.blit(texte, (100,160) ) # position du coin supérieur gauche
```

Chaque surface définit une « **bounding box** »: quand on dessine ou quand on colle dans une surface, ce qui dépasse de la bounding box n'est pas n'est pas visible (cf le cercle vert page précédente)



On peut retrouver les dimensions d'une surface et de sa bounding box:




```
largeur_texte = texte.get_width()
hauteur_texte = texte.get_height()
rect_texte = texte.get_rect()
## rect_texte == Rect(0, 0, largeur_texte, hauteur_texte)
```

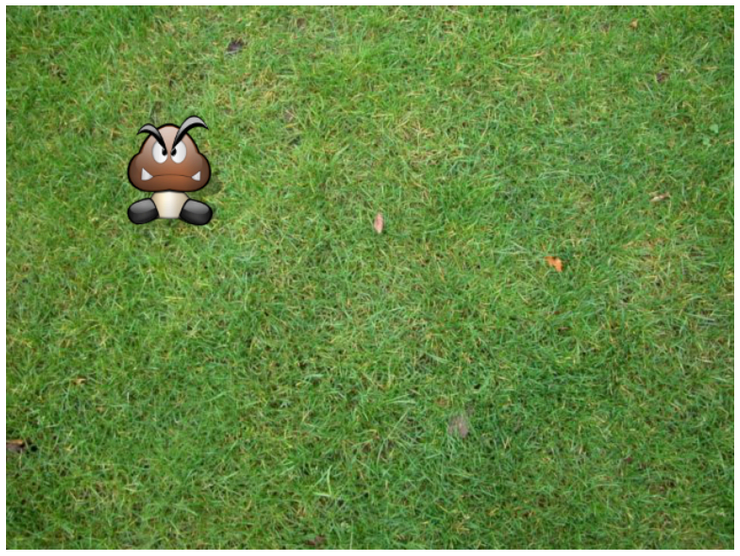
Créer une surface à partir d'une image

On peut charger un fichier contenant une image dans une surface avec la fonction load.

```
fond = pygame.image.load("gazon.jpg")
perso = pygame.image.load("perso.png")
```

Pour mettre la surface au format 8 bit, on doit ensuite appeler la méthode convert sur la surface obtenue. Si l'image contient un fond transparent, il faut utiliser la méthode convert_alpha pour ne pas perdre cette transparence.

```
(L,H) = (fond.get_width(), fond.get_height())
fenetre = pygame.display.set_mode((L,H))

fond = fond.convert() # <- après set_mode
perso = perso.convert_alpha()

fenetre.blit(fond, (0, 0))
fenetre.blit(perso, (100, 100))
```



source: openclassroom

Simuler un mouvement

- Les animations sont des techniques très utilisées dans les pages Web [par exemple avec HTML5 et son *Canvas*, ou bien Java et ses *applets*].

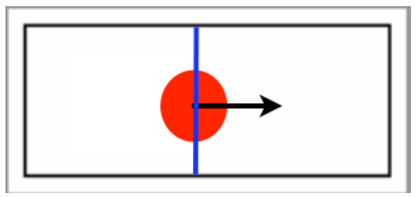
- Pygame va nous permettre de programmer facilement de petites scènes animées. Applications à la géométrie, à la physique, aux jeux, etc.

- Un **métronome** sera utilisée pour scander le temps, et donc l'évolution de la scène. On prendra le plus souvent une fréquence de 28 images par seconde (donc le métronome bat tous les 1/28e de seconde).

- Une **animation** se construit comme un **dessin animé** : ce n'est en effet pas autre chose qu'une suite d'images défilant très vite pour donner l'illusion du mouvement !

Exemple 1: un pendule horizontal

• Nous allons animer une balle rouge qui oscille entre les murs gauche et droit, avec une vitesse nulle au rebond (une sorte de *pendule horizontal*).



• Plutôt que traiter un problème de *collision* entre le disque et les murs, nous modéliserons la position du centre de la balle sous la forme d'une **fonction périodique du temps t**.

• Et quoi de mieux qu'un **sinus** pour obtenir un mouvement périodique ?

```
(x,y) = (150 + 150 * sin(pi * t) , 50)
```

• L'abscisse x varie entre 0 et 300, tandis que y est constant à 50.

• La balle met 2 secondes pour faire une oscillation complète.

Introduire des constantes pour les paramètres

Pour pouvoir modifier l'animation plus facilement, et pour pouvoir expliquer plus facilement comment elle fonctionne, il est recommandé d'introduire des constantes globales plutôt que d'utiliser des « nombres magiques ».

```
(x,y) = (150 + 150 * sin(pi * t) , 50)
```

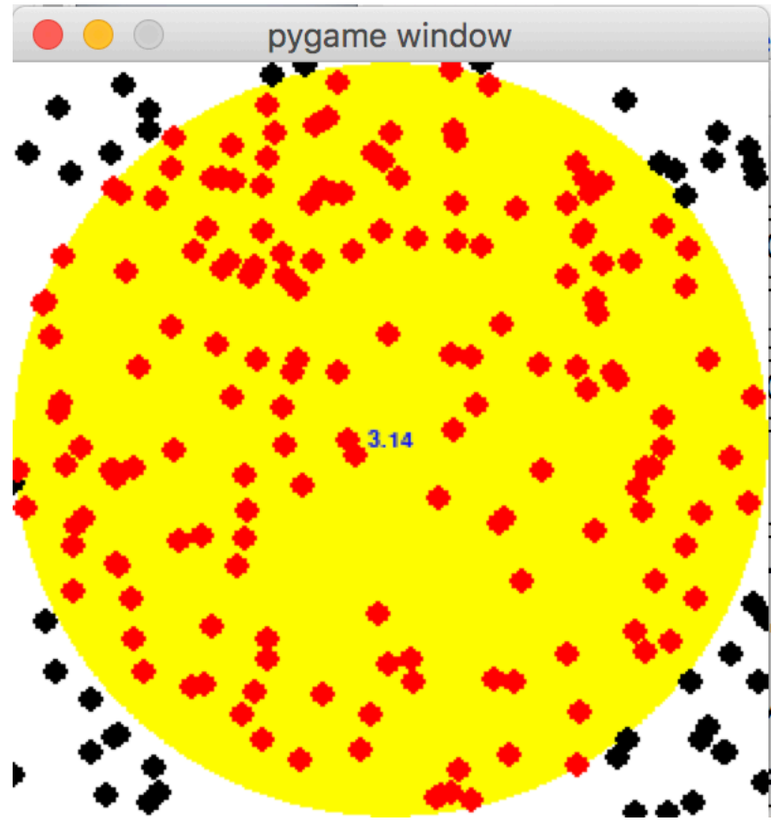
```
FREQ = 28   # la frequence d'affichage
dt = 1/FREQ # la quantite dont varie t entre deux affichages

L = 300     # largeur de la fenetre
H = 100     # hauteur de la fenetre
R = 10      # rayon de la balle

OMEGA = pi  # la vitesse angulaire du pendule

[…]

(x,y) = (L/2 + L/2 * sin(OMEGA * t) , H/2)
```

Le coeur de l'animation: la boucle infinie

```
metronome = pygame.time.Clock()  # démarre le métronome
t = 0       # t correspond au temps écoulé depuis le debut

# la boucle infinie
while True:

    # si il y a eu une demande de fermeture, on s'arrête
    if fermeture_demandee() : quitter()

    # sinon on redessine entièrement la scène
    fenetre.fill(blanc)
    (x,y) = (L/2 + L/2 * sin(OMEGA * t) , H/2)
    (x,y) = int(x), int(y)
    pygame.draw.circle(fenetre, rouge, (x,y) , R)
    pygame.display.update()

    # on incrémente le compteur de temps
    t += dt

    # on attend la pulsation pour boucler
    metronome.tick(FREQ)
```

Exemple 2: calcul de pi par lancer de fléchettes

Nous allons lancer des flèches au hasard sur un carré 2X2 contenant un disque de rayon 1.



Après n lancers, on compte le nombre f de fléchettes qui ont atteint le disque.

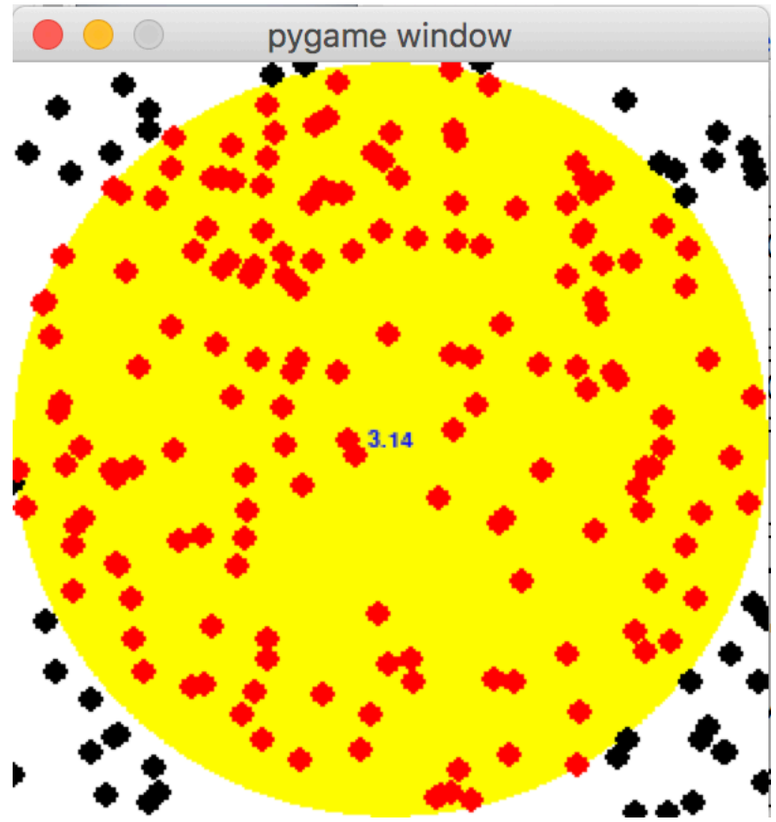
Ceci nous permet de calculer pi! En effet:

P(flèche dans le disque) = aire(disque) / aire(carre)
= pi / 4
≈ f / n lorsque n-> infini

Exemple 2: calcul de pi par lancer de fléchettes

Nous aurons besoin de savoir tirer une flèche au hasard :

```
def tire_fleche():
    x = random.randint(0, L-1)
    y = random.randint(0, L-1)
    return (x,y)
```



et nous aurons besoin de savoir si une flèche est dans le disque :

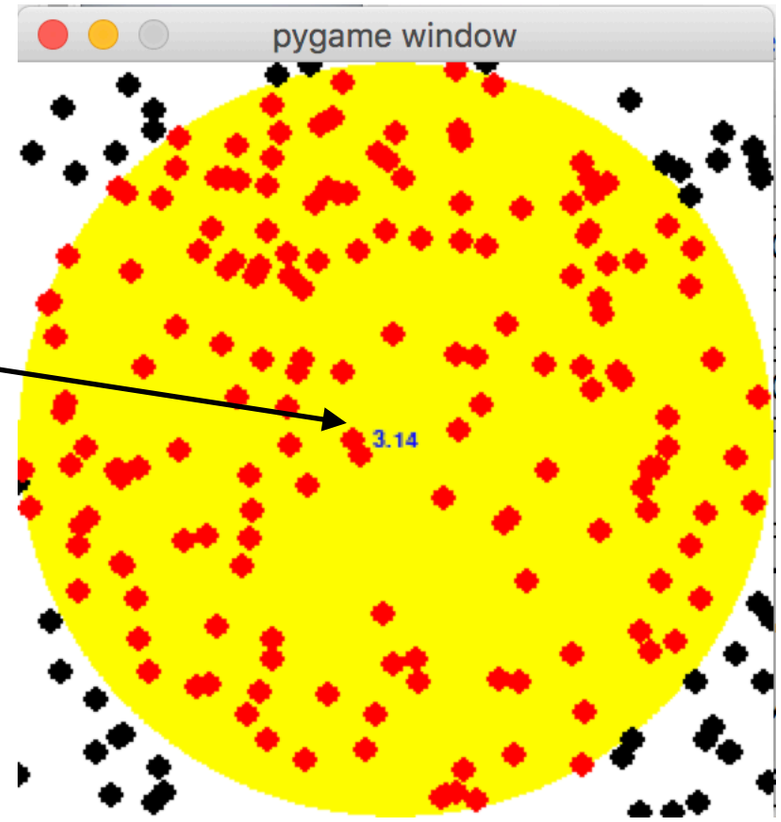
```
def dans_disque(fleche):
    (x,y) = fleche
    dist_centre = sqrt((x-L/2) ** 2 + (y-L/2) ** 2)
    return dist_centre <= L/2
```

La boucle principale, version 1

```
fenetre.fill(couleur_fond)
pygame.draw.circle(fenetre, couleur_disque, (L//2,L//2), L//2)

# la boucle principale
while True:

    # si il y a eu une demande de fermeture, on s'arrête
    if fermeture_demandee() : quitter()

    # sinon on tire une nouvelle flèche
    (x,y) = tire_fleche()

    # et on la dessine
    if dans_disque(x,y):
            pygame.draw.circle(fenetre, couleur_dedans, (x,y), R)
    else:
            pygame.draw.circle(fenetre, couleur_dehors, (x,y), R)

  pygame.display.update()


    # on attend la pulsation pour continuer
    metronome.tick(FREQ)
```

Et si on veut afficher pi?

Supposons qu'on veuille rajouter l'affichage de pi au centre de l'image



Il faut mettre à jour ce texte à chaque lancer, donc effacer le texte précédent.

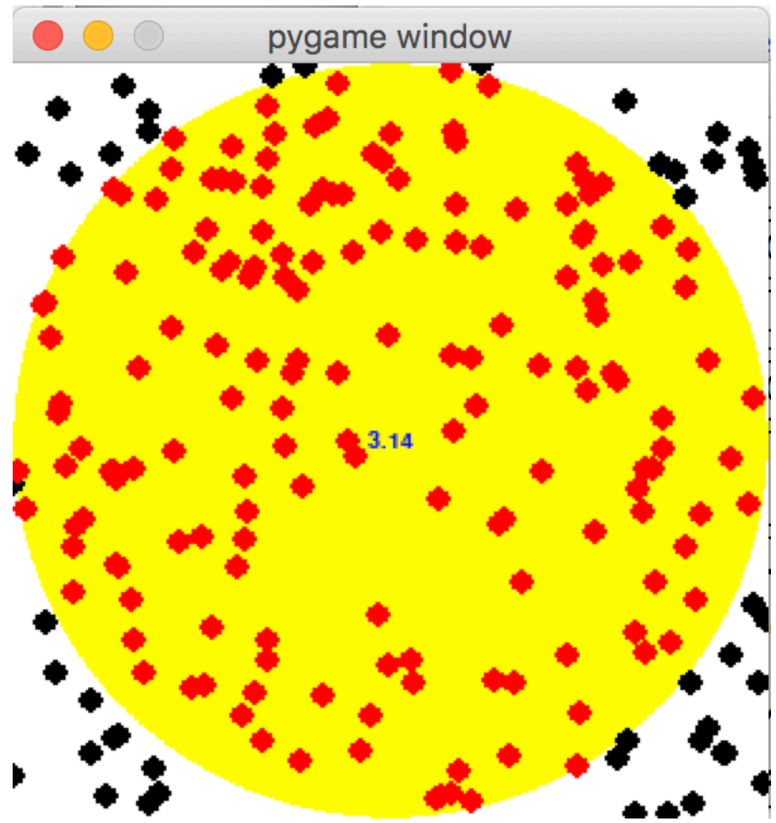
On pourrait dessiner un rectangle jaune pour effacer, mais ce n'est pas très propre.

On va au contraire recalculer l'image en entier à chaque pulsation, comme pour le pendule horizontal.

Nous allons pour cela maintenir une **liste des flèches lancées**

La fonction qui calcule le texte à afficher

```
def texte_approx_pi(fleches):

    if fleches == [] :
        approx = 0
    else :
        f = 0
        n = 0
        for fleche in fleches :
            if dans_disque(fleche) : f += 1
             n += 1
         approx = 4 * s / n

s ="{0:.2f}".format(approx)
return mafonte.render(s, True, couleur_texte)
```



Renvoie une surface qui contient le texte, qu'il faudra ensuite coller

La boucle principale, version 2

```
fleches = []    # au depart, aucune flèche n'a été lancée

# la boucle principale
while True:
    if fermeture_demandee() : quitter()

    fenetre.fill(couleur_fond) # on redessine toute la scène
    pygame.draw.circle(fenetre, couleur_disque, (L//2,L//2), L//2)
    for (x,y) in fleches :
        if dans_disque(x,y):
            pygame.draw.circle(fenetre, couleur_dedans, (x,y), R)
        else:
            pygame.draw.circle(fenetre, couleur_dehors, (x,y), R)
    texte = texte_approx_pi(fleches)
    pos_texte = (L-texte.get_width())//2, (L-texte.get_height())//2
    fenetre.blit(texte, pos_texte)
    pygame.display.update()

    # on tire une nouvelle fleche
    fleches.append(tire_fleche())

    # on attend la pulsation pour continuer
    metronome.tick(FREQ)
```

La liste des flèches s'allonge à chaque itération

Le modèle MVC (modèle-vue-contrôle)

De quoi s'agit-t-il? D'une méthodologie pour structurer son code de façon à éviter le « code spaghetti ».

---

1. Préciser le **Modèle mathématique** ou **Monde** : **l'ensemble minimum des variables qui décrivent l'état des objets.** TRES IMPORTANT !

- *la position* ***(x,y)*** *pour une balle se dirigeant au hasard.*
- *l'angle polaire* ***θ*** *pour une balle tournant sur un cercle.*
- *un paramètre* ***t*** *pour une balle se dirigeant sur une trajectoire d'équation paramétrique* $x = f(t)$, $y = g(t)$, *etc*

2. Préciser comment le Monde **évolue** à chaque top d'horloge

3. Préciser comment le Monde sera transformé en une **scène** (image rectangulaire) contenant des images : la ***Vue***.

4. Préciser (optionnellement) comment ce Monde va **interagir** avec l'utilisateur, via le clavier ou la souris.

Le modèle MVC (modèle-vue-contrôle)

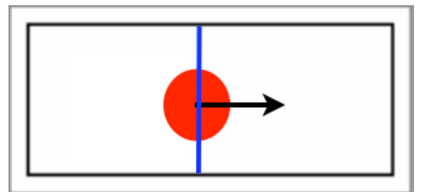
Concrètement, cela consiste à passer par une **fonction générique** boucle_principale pour réaliser son animation

```
def boucle_principale(monde_initial) :

    monde = monde_initial

    while True :

        if fermeture_demandee() : quitter()
        dessine(monde)
        pygame.display.update()
        monde = suivant(monde)
        metronome.tick(FREQ)
```

Les fonctions dessine et suivant, elles, dépendent évidemment de l'animation, de même que la notion de monde (un entier, un flottant, une liste, etc).

Le pendule horizontal, en modèle vue calcul

Le monde est un flottant!

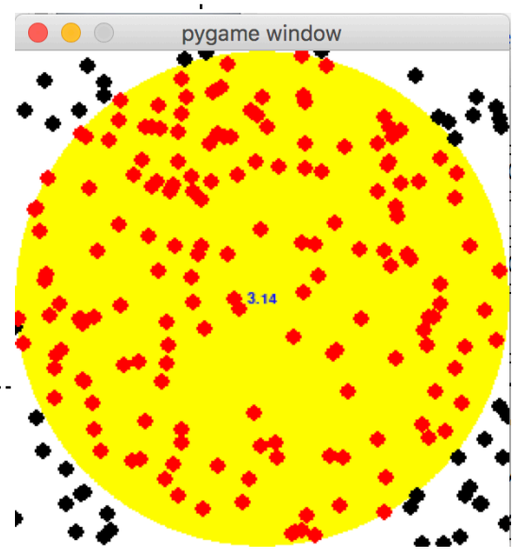


```
monde_initial = 0

def dessine(monde) :
    t = monde   # donnons-lui le nom de tout à l'heure!
    fenetre.fill(blanc)
    (x,y) = (L/2 + L/2 * sin(OMEGA * t) , H/2)
    (x,y) = int(x), int(y)
    pygame.draw.circle(fenetre, rouge, (x,y) , R)

def suivant(monde) :
    return monde + dt

def boucle_principale(monde_initial) :
  […]

# lancement de l'animation
boucle_principale(monde_initial)
```

Le lancer de flèches version 2, en modèle vue calcul

Le monde est une liste!

```
monde_initial = []

def dessine(monde) :
    fenetre.fill(couleur_fond)
    pygame.draw.circle(fenetre, couleur_disque, (L//2,L//2), L//2)
    for (x,y) in monde :
        if dans_disque(x,y):
            pygame.draw.circle(fenetre, couleur_dedans, (x,y), R)
        else:
            pygame.draw.circle(fenetre, couleur_dehors, (x,y), R)
    texte = texte_approx_pi(monde)
    pos_texte = (L-texte.get_width())//2, (L-texte.get_height())//2
    fenetre.blit(texte, pos_texte)

def suivant(monde) :
    return monde + [tire_fleche()]
```



Les évènements

L'animation se doit d'**interagir avec l'utilisateur**, ne serait-ce que pour lui permettre de mettre fin à l'animation.

Revenons à la fonction fermeture_demandee dont nous n'avons pas encore parlé.

```
def fermeture_demandee():
    for ev in pygame.event.get():
        if ev.type == QUIT :
            return True
    return False
```

La fonction **event.get()** renvoie la **liste** des évènements survenus depuis la dernière itération. Cela peut être un mouvement de la souris, une pression clavier, un redimensionnement de la fenêtre, etc. Un évènement aura donc divers attributs selon ce dont il s'agit. La fonction fermeture_demandee renvoie vrai si l'un des évènements récents a comme attribut type la constante QUIT. C'est le cas lorsque l'utilisateur a demandé à fermer la fenêtre.

Les évènements clavier

Un évènement clavier est un évènement de type KEYUP ou KEYDOWN.
Un tel évènement a aussi un **attribut** key qui permet de savoir quel bouton a été pressé ou relâché.

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

metronome = pygame.time.Clock()

print("presser q pour quitter\n")

while True:
    for ev in pygame.event.get():
        if ev.type == KEYDOWN :
            print("bouton {0} pressé\n".format(ev.key))
            if ev.key == K_q :
                exit()
        if ev.type == KEYUP:
            print("bouton {0} relâché\n".format(ev.key))
    metronome.tick(10)
```

Un espion clavier qui affiche les codes des boutons pressés ou relâchés.

Les codes sont accessibles via des **constantes** de la forme K_... (voir doc)

Le modèle MVC (modèle-vue-contrôle) avec clavier

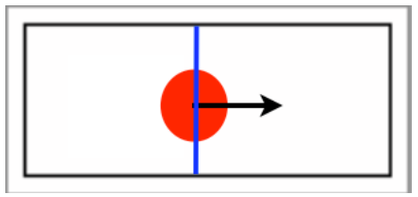
On étend la fonction boucle_principale générique. Elle appelle désormais une **fonction clavier** qui s'occupe de modifier le monde suivant les évènements claviers

```
def boucle_principale(monde_initial) :
    monde = monde_initial
    while True :
        for ev in pygame.event.get() :
            if ev.type == QUIT : quitter()
            if ev.type == KEYDOWN : monde = clavier(monde, ev.key, True)
            if ev.type == KEYUP : monde = clavier(monde, ev.key, False)
        dessine(monde)
        pygame.display.update()
        monde = suivant(monde)
        metronome.tick(FREQ)
```

Cette fonction clavier dépend de l'animation

```
def clavier(monde, bouton, presse) :
 […]
```

Exemple : pendule horizontal multicolore



```
# le monde est un couple (t,couleur)
monde_initial = (0,rouge)

def dessine(monde) :
    (t, couleur) = monde
    fenetre.fill(blanc)
    (x, y) = (L/2 + L/2 * sin(OMEGA * t), H//2)
    (x, y) = int(x), int(y)
    pygame.draw.circle(fenetre, couleur, (x,y) , R)
    pygame.display.update()


def tire_couleur():
    return
randint(0,255),randint(0,255),randint(0,255)


def clavier(monde, bouton, presse):
    (t, _) = monde
    if presse :
        if bouton == K_F1 :
            return (t, rouge)
        return (t, tire_couleur())
    return monde
```

A chaque pression d'une touche clavier, on tire une nouvelle couleur au hasard. La touche F1 réinitialise à la couleur rouge par défaut.

```
def suivant(monde) :
    (t, c) = monde
    return (t+dt, c)
```

Les évènements souris

• **Gestion de la Souris.** La communication d'un programme avec l'utilisateur se fait principalement à travers le clavier et la souris. Quid de la souris ?

• La pression d'une touche du clavier gérait un ***évènement-clavier***.

• Quels sont les ***évènements-souris*** ?

Ils sont de différents types:

- ev.type == MOUSEMOTION : la souris a bougé
- ev.type == MOUSEBUTTONDOWN : un bouton souris a été pressé
- ev.type == MOUSEBUTTONUP : un bouton souris a été relâché

On peut alors traiter l'évènement en appelant les fonctions

- pygame.mouse.get_pos() -> (x,y) la position de la souris
- pygame.mouse.get_pressed() -> (b1,b2,b3) les booléens pour chaque bouton

Exemple : pendule horizontal multicolore

Réagissons aussi à la souris

- le bouton gauche change la couleur au hasard
- le bouton droit la remet à la couleur initiale

```
def souris(monde, type) :
    (t, _) = monde
    if type == MOUSEBUTTONDOWN :
        if pygame.mouse.get_pressed()[0] :
            return (t, tire_couleur())
        return (t, rouge)
    return monde

def boucle_principale(monde_initial) :
    monde = monde_initial
    while True:
        for ev in pygame.event.get() :
            […]
            if ev.type in [MOUSEMOTION, MOUSEBUTTONDOWN, MOUSEBUTTONUP] :
                monde = souris(monde, ev.type)
        […]
```