POLYTECH PARIS-UPMC

Université Pierre et Marie Curie
2016-2017

MAIN - Python

Travaux Pratiques

— TP4 —

Objectif(s)

- Création d'une classe et d'instances de classe
- Écriture et utilisation de méthodes
- Utilisation du module `pyGame`

Le but de ce TP est de créer un jeu de billard simplifié (15 boules jaunes et rouges, 1 boule blanche, une table, aucun trou), qui servira simplement de prétexte pour modéliser des boules qui s'entrechoquent.

Ce TP utilisera les fichiers image qui se trouvent à l'adresse `/home/sasl/shared/main3/python/TP4`, ainsi que les fichiers `boule.py` et `affichage.py` qui contiennent une version incomplète du programme final.

L'affichage de la table et des boules de billard, leur animation, et l'utilisation de la souris seront réalisés grâce au module `PyGame`[1]. Il sera important de se reporter à la documentation (`http://www.pygame.org/docs/ref/`) afin de connaitre le prototype des fonctions/méthodes que l'on vous demandera d'utiliser.

Question 1 – Méthodes `__init__` et `__str__` de la classe `Boule`

Le fichier `boule.py` contient un squelette de la classe `Boule` qui permettra de créer des objets représentant des boules de billard. Les prototypes des méthodes sont fournis. Vous devrez écrire le code manquant.
Une boule sera représentée par son vecteur position `(x,y)` et son vecteur vitesse `(vx,vy)`. Le rayon sera le même pour toutes les boules et est stocké dans une variable nommée `RAYON` donnée en tout début du fichier `boule.py`.

1. Complétez la méthode `__init__` pour donner les bonnes valeurs aux attributs représentant la position et la vitesse.
2. Pour afficher une image, nous utiliserons les fonctions du module `PyGame`. Il est tout d'abord nécessaire de charger l'image, puis de la mettre à la bonne échelle :
   - Utilisez la fonction `pygame.image.load` afin de charger l'image (jpg, png, etc.) dont le nom est donné en paramètre de la méthode `__init__`. Cette fonction renvoie un objet *surface* que l'on pourra utiliser ultérieurement. Attention, si l'image représente une boule, elle a tout de même les dimensions d'un carré. Consultez le fichier `boule_blanche.png` par exemple ; c'est un carré de 30 pixels par 30 pixels dans lequel le disque représentant la boule est tangent aux côtés.
   - Utilisez la fonction `pygame.transform.scale` qui accepte deux arguments (une surface et un tuple de deux entiers indiquant les nouvelles largeur et longueur) afin de modifier les dimensions de la surface qui vient d'être récupérée de manière à ce que le disque représentant la boule ait un rayon égal à la constante `RAYON`. L'image finale sera stockée dans l'attribut nommé `image`.
3. Complétez la méthode `__str__` dont le prototype est donné. Cette méthode devra afficher joliment les informations sur la position et la vitesse de la boule.
4. Testez ces deux méthodes. Pour cela, créez un fichier nommé `test.py` dans lequel vous importerez la classe `Boule` du module `boule` que vous venez de compléter, afin de pouvoir créer des boules. Écrivez un code qui permette de créer une boule, à laquelle vous donnerez une position et une vitesse de votre choix, et affiche dans le terminal les informations relatives à cette boule.

[1] `www.pygame.org/`

Pour le moment, nous n'avons utilisé l'attribut `image` dans aucune méthode. Dans la question qui suit, nous allons utiliser le package `pygame` afin d'afficher l'image d'une boule.

Question 2 – Affichage graphique d'une boule - Premiers pas avec `pyGame`

Tout l'affichage sera géré par le module `pyGame` qui est un module spécialisé pour écrire facilement de petits jeux graphiques en python (avec animations, sons, gestion de la souris, clavier, *etc.*).
L'utilisation de `pyGame` se fait de la manière suivante :

a) on importe le module `pyGame`

b) on initialise l'affichage avec

```
pygame.init()
```

c) on crée une fenêtre de jeu avec la commande

```
ecran = pygame.display.set_mode( dimension )
```

où `dimension` est un tuple de deux entiers (largeur et hauteur en pixels, ici 500 et 800)

d) on peut définir le titre de la fenêtre avec la fonction `pygame.display.set_caption` qui prend comme argument le nom à donner à la fenêtre (chaine de caractères)

e) on peut charger une image afin de la coller sur la fenêtre et ainsi avoir un joli fond

```
fond = pygame.image.load("joli_fond.png")
```

f) enfin, il faut créer une boucle dans laquelle on doit traiter les évènements (clics souris, touches au clavier, etc...) et rafraîchir l'affichage :

```
# boucle de gestion des événements et d'affichage (dont on ne sort que si l'on
    ferme la fenêtre)
onContinue = True
horloge = pygame.time.Clock()
while onContinue:

    # Gestion des événements (on parcourt tous les événements, et si un évènement
        est de type QUIT, alors on ne continue plus)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            onContinue = False

    # on affiche ici ce que l'on a à afficher
    ecran.blit(fond, (0,0)) # on colle le fond en position (0,0) qui sont les
        coordonnées du coin supérieur gauche de la fenêtre
    ...             # on colle d'autres images par-dessus le fond

    # on met à jour l'affichage
    pygame.display.flip()

    # on attend un petit peu, pour ne boucler que 25 fois maxi par seconde
    horloge.tick(25)
```

1. Utilisez la description précédente pour compléter le fichier `affichage.py`. Même si c'est parfois une simple recopie qui est demandée, le but est de bien comprendre ce que vous écrivez.
2. Utilisez l'image `billard.png` comme image de fond.
3. Utilisez la méthode `blit` pour compléter la méthode `affichage` de la classe `Boule`. La méthode `blit` s'applique sur une surface et accepte deux arguments : une autre image (surface) à coller, et un tuple de deux entiers indiquant les coordonnées où coller cette seconde image.
4. Créez une boule test avec une position initiale `(0,0)` (la vitesse n'a pas d'importance puisque nous n'avons pas encore créé de méthode pour faire bouger une boule).

5. Utilisez sa méthode d'affichage pour l'afficher sur le billard. Le centre de la boule affichée se trouve-t-il effectivement dans le coin supérieur gauche de la fenêtre ? Si oui, vous n'êtes pas concerné(e) par les deux questions qui suivent.

6. Sinon déduisez-en la translation à appliquer dans la méthode `affichage` de la classe `Boule` afin que l'image de la boule ait son centre effectivement situé aux coordonnées conservées en attribut.

7. Testez à nouveau l'affichage d'une boule avec un centre situé en `(0,0)` (qui correspond aux coordonnées du coin supérieur gauche de la fenêtre) afin de vérifier que votre méthode `affichage` est correcte.

Question 3 – Déplacement des boules

Complétez le code de la méthode `deplace` de la classe `Boule` qui calcule les nouvelles coordonnées de la boule à partir des anciennes et de la vitesse. Nous approximerons l'équation qui nous donne la nouvelle position x par $x \leftarrow x + \Delta x$ et $vx = \frac{\Delta x}{\Delta T}$, avec ΔT une petite constante de temps (déjà fixée dans la variable `DELTA_T` fournie). Idem pour y.

Afin de vérifier que votre solution est correcte, créez une boule test en position `(RAYON,RAYON)` avec une vitesse initiale `(5,8)`. N'oubliez pas de faire déplacer la boule avant chaque nouvel affichage.

Si tout se passe bien, la boule doit traverser le billard en diagonale. Plus précisément, à chaque nouvelle itération de la boucle principale de `pyGame`, on calcule la nouvelle position de la boule grâce à la fonction `deplace`, puis on affiche son image à cette nouvelle position.

Vous remarquerez que la boule continue son chemin et disparaît de la fenêtre. L'objectif de la question suivante est donc de gérer les rebonds sur les bandes de la table de billard.

Question 4 – Rebonds sur les bandes de la table

1. Complétez le code de la méthode `rebond` qui fonctionne de la manière suivante :

   - si la boule est à distance inférieure ou égale à son rayon d'un bord vertical de la fenêtre, la composante horizontale de la vitesse change de signe ;
   - si la boule est à distance inférieure ou égale d'un bord horizontal de la fenêtre, la composante verticale de la vitesse change de signe ;
   - dans les deux cas, la boule est, s'il le faut, replacée de manière tangente au bord afin de ne jamais afficher une boule qui dépasserait les limites.

2. Modifiez légèrement cette méthode pour que les rebonds ne s'effectuent plus sur les bords de la fenêtre mais sur les bandes de la table de billard, sachant que la largeur de ces bandes est stockée dans la variable `BORD`.

3. Testez cette nouvelle méthode !

Question 5 – Gestion des collisions entre plusieurs boules

Nous cherchons ici à gérer la collision entre deux boules afin qu'elles puissent s'entrechoquer.

1. Pour gérer la collision entre deux boules, nous devons tout d'abord savoir s'il y a collision ! Pour cela, complétez la méthode `dist` qui prend en paramètre une boule et retourne la distance euclidienne centre à centre entre les deux boules.

2. Nous considérons que toutes les boules ont même masse, et que les collisions sont élastiques. Pour cela, il sera plus facile de se placer dans une base différente de la base orthonormée $\mathscr{B}_1 = (\vec{u}_x, \vec{u}_y)$ dans laquelle nous avons exprimé jusqu'à maintenant les vitesses des boules. Ainsi, nous allons nous doter d'une deuxième base orthonormée $\mathscr{B}_2 = (\vec{n}, \vec{g})$. Les vecteurs $\vec{n}$ et $\vec{g}$ sont illustrés Fig. 1.

   Une fois que nous aurons calculé cette nouvelle base, la démarche sera la suivante :

   (i) se placer dans un contexte où les deux boules sont tangentes l'une à l'autre. Plus précisément, lors de la détection d'une collision, à cause de la discrétisation des déplacements, nous pourrons nous retrouver dans le cas illustré par la Fig. 2. La démarche à adopter sera donc de déplacer chacune des deux boules d'une quantité $\frac{\delta}{2}$ dans la direction de $\vec{n}$ mais bien évidemment dans des sens opposés.

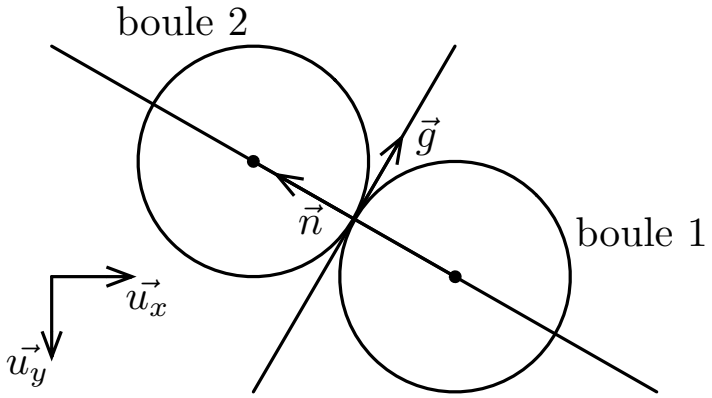

Figure 1: Base orthonormée $(\vec{n},\vec{g})$ associée au choc entre une boule 1 et une boule 2

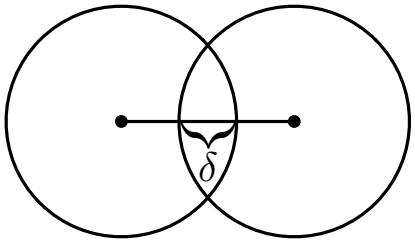

Figure 2: Cas de superposition de la boule 1 avec la boule 2

(ii) exprimer les vitesses des deux boules dans $\mathscr{B}_2$, échanger leurs composantes normales (*i.e.* suivant $\vec{n}$), et réexprimer les vitesses dans la base $\mathscr{B}_1$:

Notons $\vec{p}_1$ le vecteur position de la boule 1, donné par ses coordonnées $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}_{\mathscr{B}_1}$ dans la base $\mathscr{B}_1$. Idem pour la boule 2. Alors nous obtenons facilement $\vec{n}$ par :

$$\vec{n} = \frac{\vec{p}_2 - \vec{p}_1}{\|\vec{p}_2 - \vec{p}_1\|}.$$

De cette manière nous obtenons simplement les coordonnées de $\vec{n}$ dans $\mathscr{B}_1$, que nous noterons (n_x, n_y). Ensuite, nous en déduisons les coordonnées (g_x, g_y) de $\vec{g}$ dans $\mathscr{B}_1$ en utilisant simplement le fait que $\vec{n}.\vec{g} = 0$ (le choix de l'orientation de la base $(\vec{n},\vec{g})$ n'a aucune importance ici).

À ce stade, la base $(\vec{n},\vec{g})$ est construite. Nous allons maintenant exprimer les vecteurs vitesse dans cette base. Notons, pour $i = 1,2$, $\vec{v}_i = \begin{pmatrix} v_{i,x} \\ v_{i,y} \end{pmatrix}_{\mathscr{B}_1} = \begin{pmatrix} v_{i,n} \\ v_{i,g} \end{pmatrix}_{\mathscr{B}_2}$ les coordonnées des vecteurs vitesse des deux boules dans les bases $\mathscr{B}_1$ et $\mathscr{B}_2$ respectivement. On rappelle que la matrice de changement de base pour passer de la base $\mathscr{B}_1$ à la base $\mathscr{B}_2$ est la suivante :

$$P_{1\to 2} = \begin{pmatrix} \vec{u}_x.\vec{n} & \vec{u}_y.\vec{n} \\ \vec{u}_x.\vec{g} & \vec{u}_y.\vec{g} \end{pmatrix}$$

Comme les deux bases sont orthonormées, $P_{1\to 2}$ est une matrice orthogonale. Son inverse est donc sa transposée et on a :

$$P_{2\to 1} = P_{1\to 2}^{\top} = \begin{pmatrix} \vec{u}_x.\vec{n} & \vec{u}_x.\vec{g} \\ \vec{u}_y.\vec{n} & \vec{u}_y.\vec{g} \end{pmatrix}$$

Une fois ces outils à disposition, on passe simplement d'une représentation à l'autre par :

$$P_{1\to 2} \times \begin{pmatrix} v_{i,x} \\ v_{i,y} \end{pmatrix}_{\mathscr{B}_1} = \begin{pmatrix} v_{i,n} \\ v_{i,g} \end{pmatrix}_{\mathscr{B}_2}, \; P_{2\to 1} \times \begin{pmatrix} v_{i,n} \\ v_{i,g} \end{pmatrix}_{\mathscr{B}_2} = \begin{pmatrix} v_{i,x} \\ v_{i,y} \end{pmatrix}_{\mathscr{B}_1}$$

Lors du choc, le seul changement à appliquer aux deux vitesses est un échange de leur composante normale (les composantes tangentielles restent inchangées) : $\vec{v}_1$ devient le vecteur défini par les coordonnées $\begin{pmatrix} v_{2,n} \\ v_{1,g} \end{pmatrix}_{\mathscr{B}_2}$ dans la base $\mathscr{B}_2$, et $\vec{v}_2$ le vecteur défini par les coordonnées $\begin{pmatrix} v_{1,n} \\ v_{2,g} \end{pmatrix}_{\mathscr{B}_2}$ dans la base $\mathscr{B}_2$. Il ne reste alors plus qu'à réexprimer ces nouveaux vecteurs vitesse dans la base $\mathscr{B}_1$.

Utilisez ces informations pour compléter le code de la méthode `collision` de la classe Boule. Ensuite, testez-la en créant deux boules avec des vitesses non nulles et choisies pour qu'une collision soit possible... rapidement...

Attention : la méthode `collision` ajuste la vitesse de la boule sur laquelle on appelle la méthode, ainsi que la boule passée en argument de la méthode. Ainsi, si on fait `boule1.collision(boule2)`, on ne devra pas faire `boule2.collision(boule1)`, *i.e.* on ne doit pas gérer deux fois une même collision.

Question 6 – Mise en place des boules pour une partie de billard

Le but de cette question est de créer 16 boules et les placer dans la configuration de la Fig. 3. Pour cela, vous disposez d'un fichier nommé `info_boules.txt`. Ce fichier contient toutes les coordonnées des 16 boules à créer, ainsi que leurs couleurs. Par exemple, la première ligne est `172,67,boule_rouge.png`. Ainsi, nous devrons créer une boule rouge avec une vitesse initiale nulle, une position initiale égale à `(172,67)`, et qui utilisera le fichier image `boule_rouge.png` pour s'afficher.

Dans le fichier `affichage.py` est donné le prototype d'une fonction nommée `position_initiale` qui crée une liste contenant 16 boules en récupérant les données contenues dans le fichier `info_boules.txt`, et renvoie la liste. Ces boules auront une vitesse initiale nulle. Écrivez le code de cette fonction.

Dans le programme principal, vous utiliserez cette fonction pour créer les boules en configuration initiale. Vous conserverez la liste renvoyée par la fonction `position_initiale` dans une variable nommée `listeBoules`. Vous comprendrez aisément qu'il est également conseillé, pour la suite, de conserver la boule blanche non seulement dans la liste `listeBoules` mais aussi dans une variable nommée `BB` par exemple, afin d'y avoir facilement accès.

Question 7 – Une fonction pour faciliter la gestion des collisions du stock de boules

Dans le fichier `affichage.py` est donné le prototype d'une fonction nommée `gestion_collisions`. Cette fonction prend en paramètre une liste de boules, et utilise la méthode `collision` de la classe Boule afin de gérer l'ensemble des collisions entre toutes les boules de la liste pour `gestion_collisions`.

Complétez le code de cette fonction. Cette fonction devra veiller à ne pas gérer deux fois une même collision, comme indiqué en fin de question 5.

Une fois chose faite, testez cette fonction. Pour cela vous pourrez utiliser la fonction `position_initiale` créée précédemment en la modifiant légèrement de manière à imprimer une vitesse non nulle à la boule blanche (par exemple `(0,-60)`).

Les méthodes `affiche`, `deplace` et `rebond` seront appliquées à chaque boule en utilisant une simple boucle `for`, dans la boucle `while` principale pour l'affichage. À chaque itération de la boucle `while`, nous appliquerons la méthode `deplace` et `rebond` à toutes les boules. Puis nous utiliserons la fonction `gestion_collisions`. Et enfin nous afficherons toutes les boules via leur méthode `affiche`.

Question 8 – Tirer la boule blanche !

Nous allons compléter notre programme afin de pouvoir modifier le vecteur vitesse de la boule blanche juste avec un clic gauche de souris.

Ce nouveau vecteur vitesse sera par exemple la moitié du vecteur ayant pour origine le centre de la boule blanche et pour extrémité la position de la souris au moment du clic. Vérifiez que votre code fonctionne.

Question 9 – Se replacer en configuration initiale

De la même manière que précédemment, nous allons utiliser le clic droit de la souris pour effectuer une nouvelle action. Dès que l'utilisateur cliquera droit, un retour à la configuration initiale devra être fait (on pourra utiliser la fonction `position_initiale()` ; veillez bien à restocker la boule blanche dans votre variable `BB`).

Figure 3: Configuration des boules

Question 10 – Forces de frottement

Vous l'avez remarqué, nos boules ne s'immobilisent pas. Pour rendre le comportement des boules plus réaliste, nous allons introduire l'effet d'une force de frottement qui, pour une boule ayant un vecteur vitesse $\vec{v}$ à un instant t, est égal à $\vec{F} = -\mu_f \frac{\vec{v}}{\|\vec{v}\|}$, où $\mu_f > 0$ est une constante commune à toutes les boules. Ainsi, en considérant le principe fondamental de la dynamique et en supposant que cette force de frottement est la seule force à composante horizontale non nulle s'exerçant sur les boules, cela implique que les boules en mouvement subissent une accélération $\vec{a} = \frac{\vec{F}}{m}$ constante, où m est la masse d'une boule. La norme de cette accélération constante subie par les boules en mouvement est donnée dans la variable nommée `ACC` (qui dépend donc de la masse et de μ_f).

Complétez la méthode `calculeVitesse` de la classe `Boule`. Cette méthode fait deux choses :

- si la norme du vecteur vitesse est inférieur à la constante `EPSILON`, alors le vecteur vitesse est mis à zéro ;
- sinon, la vitesse diminue à cause de l'accélération subie de par la force de frottement. Le vecteur vitesse $\vec{v}$ devient $\vec{v} - ACC * DELTA\_T * \frac{\vec{v}}{\|\vec{v}\|}$.

Utilisez cette méthode dans la boucle `pyGame`, et vérifiez son bon fonctionnement.

Vous pouvez vous amuser à modifier les valeurs des constantes `ACC` et `EPSILON` pour trouver un comportement qui vous semblerait plus réaliste.
Admirez enfin le résultat !!