

# PROJET - ISN

 python™ 2018-2019

PONG – REMASTERISE

Dimitrii Benoit

# PING PONG – PYTHON-PYGAME

## Table des matières

I.	Introduction et choix du projet.....	3
II.	Travail de groupe.....	3
	a. liste des différents composants nécessaire.....	3
	b. Répartition des tâches.....	4
III.	Développement de la partie travaillé.....	5
	a. Réflexion.....	5
	<i>Qu'est-ce qu'une « class » ?</i> .....	5
	b. l'environnement de travail.....	6
	c. la Balle.....	6
	d. les plateformes.....	8
	e. la mise a jour des images au cours du jeu.....	9
IV.	Conclusion.....	10
V.	les différents sites utilisés.....	10

## **I. Introduction et choix du projet**

Cette année il nous a été demandé de produire un projet pour notre BAC d'Isn. Mon groupe étant constitué de 3 personnes : Ashfaq Assenjee, Antoine Nurbel et moi, avons longuement hésité sur le choix du projet à réaliser. Ainsi après plusieurs semaines de réflexions mais aussi de documentation sur les différents modules de python possible, nous nous sommes penchés sur pygame un module rassemblant beaucoup de fonctionnalités elles même présentent dans différents modules de python.

On s'était alors mis d'accord sur la production d'un des premiers jeux d'arcade. Créé en 1972 par Allan Alcorn, avec le soutien de la compagnie ATARI, ce jeu consiste en un Ping Pong électronique en deux dimensions. Cependant le jeu original étant trop basique nous avons décidé de fusionner notre créativité pour obtenir un résultat dont nous sommes fiers d'être les créateurs.

Ainsi, pour faire court notre jeu se déroule en une manche de douze points accompagnés d'une musique nous immergeant dans un univers différent, celui du « Pong ».

## **II. Travail de groupe**

### **a. liste des différents composants nécessaire**

Nous avons dès lors, après notre choix commencé à chercher comment le reproduire à notre façon. Nous nous sommes ainsi intéressés sur le différent composant nécessaire à la création de ce jeu (composant présent dans le jeu et non un composant matériel). Nous avons alors noté que le jeu devait comprendre les éléments ci-dessous :

- Une fenêtre à dimension bien réfléchi où nous disposerons notre jeu et les différentes interfaces possibles.
- Des délimitations pour le terrain de jeu.
- Deux plateformes mobiles pouvant être contrôlées par deux joueurs à l'aide d'un clavier et des touches : Flèche gauche et droite, « A » et « D ».
- Une balle mobile dans le terrain rebondissant sur les côtes des espaces délimités.
- Des conditions de collisions entre balle et plateforme.

- Un menu constitue de trois boutons : **JOUER, AIDE** et **QUITTER**
- Un tableau de score.
- Les différentes images nécessaires à notre menu et notre tableau de score pour reproduire un climat retro lors des sessions de jeux
- Une musique de fond avec un bouton pouvant couper le son ou le relancer

### **b. Répartition des taches**

Ensuite, nous nous sommes repartis les taches de façon stratégique pour gagner le plus de temps possible. Nous avons alors utilisé des plateformes de collaboration électronique telle que celle disponible gratuitement à l'école : étant One note via Office 365, mais aussi Dropbox car parfois nos fichiers échangés étaient trop lourds pour les télécharger sur la plateforme d'échange surtout que nous avons décidé pour faciliter la compréhension du scripte d'enregistrer des audio concernant nos différentes parties.

**Nous avons alors reparti les travaux de cette façon :**



<b><u>MEMBRE DU GROUPE</u></b>	<b><u>TACHES A EFFECTUER</u></b>
Ashfaq Assenjee	Musique, Boucle principale du Jeu, Tableau de score, Fonction affichant un texte, Fonction mouvement
Antoine Nurbel	Création des images avec Photoshop, Fonction affichant un texte, Fonction mouvement, Le menu du jeu
Dimitrii BENOIT (moi)	Couleur, Fonction réinitialisant la partie, La balle mobile, Les différentes plateformes(raquettes)

### III. Développement de la partie travaillé

#### a. Réflexion

Ainsi, mon travail pour la réalisation de ce projet était destiné à la création des deux éléments les plus importants du jeu : la balle, mais aussi les deux plateformes contrôlées par les joueurs. Pour cela, il nous fallait gagner le maximum de place dans notre code.

C'est alors qu'après lecture de multiples blogs et visionnages de plusieurs heures de tutoriels sur l'utilisation de pygame dans python que j'ai trouvé la solution à notre problème ou<- beaucoup de passionnés avaient conseillé l'utilisation de « class » surtout pour les éléments mobiles présents en plusieurs exemplaires dans le script. Ces « class » nous faisaient non seulement éviter un code trop long, mais nous facilitaient aussi en ce qui concernait, les différentes modifications possibles au cours du temps dans les différentes boucles infinies cruciales pour le bon fonctionnement du jeu (fonction **Game()** et **Menu()** dans le programme).


#### *Qu'est-ce qu'une « class » ?*

- ➔ Tout d'abord, pour créer une classe nous devons utiliser le mot clef : **class** suivi du **nom de la fonction class** que nous voulons créer : 

```
49 class Ball:
```
- ➔ Toutes les classes ont une fonction appelée **`__init__`** (indenté dans cette dernière), qui est toujours exécutée lorsque la classe est lancée. Ainsi, le but de la fonction **`__init__`** est d'affecter des valeurs aux propriétés de l'objet ou à d'autres opérations nécessaires lors de la création de l'objet. Comme par exemple le paramètre « **self** » qui est une référence à l'instance actuelle de la classe et permet d'accéder aux variables appartenant à la classe. Il ne doit pas nécessairement s'appeler self, vous pouvez l'appeler comme bon vous semble, mais ce doit être le premier paramètre de n'importe quelle fonction de la **class**.

```
50     def __init__(self, x, y, size, color):
```

## b. L'environnement de travail

De plus, pour coder ce jeu, j'ai voulu avoir un environnement adéquat et confortable. Pour ce faire j'ai parcouru de nombreux sites pour trouver un éditeur avec les fonctionnalités que je cherchais. J'ai ainsi opté pour l'éditeur de texte **ATOM**, car il y avait possibilité  d'installer des plugins pour ainsi améliorer cette facilité à coder comme par exemple le fait d'avoir des propositions pour compléter automatiquement une ligne avec des variables déjà présentes dans le programme.


## c. la Balle

Ainsi à l'aide de cette classe ci-dessous j'ai pu créer une balle mobile dans un espace délimité :

```
25 RED = (255, 0, 0)
26 BLUE = (0, 0, 255)
27 GREEN = (0, 255, 0)
28 WHITE = (255, 255, 255)
29 BLACK = (0, 0, 0)

49 class Ball:
50     def __init__(self, x, y, size, color):
51         self.x = x
52         self.y = y
53         self.size = size
54         self.color = color
55         self.xx = 1
56         self.yy = 1
57         self.vel = 4
```

Comme dit précédemment, cette « class » contient la fonction \_\_init\_\_(), qui aide à l'initialisation des différentes variables de la balle. Ici respectivement la présence d'un x, y, size et couleur. On va ensuite faire l'initialisation des variables avec le paramètre « **self** ». Les deux premières variables représentent ainsi l'abscisse et l'ordonnée de la balle, puis la variable **size** représentant le rayon du cercle. La variable **color** représentant la couleur de la balle en code **(R,V,B)**. Les variables **xx** et **yy** représentant le sens de la balle. La variable **vel** définissant la vitesse de cette dernière.

Pour continuer dans chaque « class » de notre jeu nous y avons incorporé une fonction **draw()** qui dessine notre forme géométrique ici : Un cercle , à l'aide de la fonction du module pygame : `pygame.draw.circle(emplacement, couleur, (abscisse, ordonnée), rayon, surface)`

```
def draw(self, win):
    pygame.draw.circle(win, self.color, (self.x, self.y), self.size, 0)
```

Cependant pour que cette balle puisse bouger, j'ai dû déterminer les conditions ci-dessous. Celles entre la balle et les côtés **gauche** et **droit** du terrain de jeu, mais aussi les conditions de collisions entre la balle et **les deux plateformes**. Tout cela en jouant avec les coordonnées de la balle.

```
if self.x == 500 - self.size:
    self.xx = -1
if self.x == self.size:
    self.xx = 1
if self.x >= plat2.x - 10 and self.x <= plat2.x + plat2.width + 10 and self.y == plat2.y - 10:
    self.yy = -1
if self.x >= plat1.x - 10 and self.x <= plat1.x + plat1.width + 10 and self.y == plat1.y + plat1.height + 10:
    self.yy = 1
self.x += self.vel * self.xx
self.y += self.vel * self.yy
```

De plus pour que cette balle puisse bouger, il avait été alors nécessaire de modifier les coordonnées de cette dernière. C'est-à-dire en **lui rajoutant sa vitesse constante** dans toute la durée du programme a **x** et de **y**.

Cependant, pour déterminer le sens de la balle (droit,gauch,haut,bas) il a été nécessaire de créer une variable **xx** et **yy** qui multiplié à la vitesse détermine son sens :

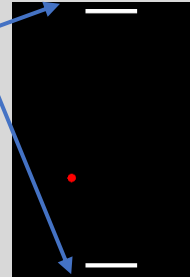
- ➔ Si **xx** = 1 et **yy** = 1, la balle ira du côté **droit** vers le **haut**.
- ➔ Si **xx** = -1 et **yy** = 1, la balle ira du côté **gauche** vers le **haut**.
- ➔ Si **xx** = 1 et **yy** = -1, la balle ira du côté **droit** vers le **bas**.
- ➔ Si **xx** = -1 et **yy**= -1, la balle ira du côté **gauche** vers le **bas**.

Ainsi, pour déterminer quand est-ce qu'un point est marqué, mais aussi pour réinitialiser la position de la balle et relancer un autre point j'ai alors décidé de créer cette fonction ci-dessous pour faciliter cela et pour avoir un gain de place :

```
31 sens = [-1, 1]
41 def reset_ball(index):
42     index.y = 325
43     index.x = 250
44     index.xx = random.choice(sens)
45     index.yy = random.choice(sens)
46     index.vel = 4
61
62     if self.y >= 650 - self.size:
63         reset_ball(self)
64         plat1.score += 1
65
66     if self.y <= self.size:
67         reset_ball(self)
68         plat2.score += 1
```

La fonction `reset_ball()` remet la balle à sa position initiale et lui donne un sens aléatoire en piochant à l'aide du module random, dans la liste `sens[1,-1]` un `xx` et un `yy`.

De plus, pour déterminer quand est-ce que cette fonction doit être appelée et quand est-ce qu'un joueur score un point, nous avons créé des conditions `if`, déterminant les limites se situant à l'arrière de chaque plateforme (raquette) :



Ainsi avant de pouvoir insérer notre balle dans notre jeu nous l'avons définie dans une variable : `97 PongBall = Ball(250, 325, 10, RED)`.

#### d. les plateformes

Nous avons alors utilisé la même méthode pour les plateformes(raquettes) :

```
81 class platform:
82     def __init__(self, x, y, width, height, color):
83         self.x = x
84         self.y = y
85         self.width = width
86         self.height = height
87         self.color = color
88         self.vel = 5
89         self.score = 0
```

Cette « class » est alors constitué de plusieurs variables qui sont les suivantes :

Ses coordonnées `x` et `y`, la variable `width` représentant sa largeur, la variable `height` représentant sa hauteur, et la variable `color` représentant sa couleur en code (`R`, `V`, `B`), la variable `vel` représentant sa vitesse et la variable `score` représentant le score personnel de la plateforme pour ainsi pouvoir l'afficher sur le tableau de score.

Pour continuer, il y a aussi la présence d'une fonction `draw()` dessinant notre forme



rectangulaire à l'aide de la fonction : `pygame.draw.rect(emplacement, couleur, (abscisse, ordonnée, largeur, hauteur))`.

```
91     def draw(self, win):
92         pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.height))
```

Enfin, nous avons défini deux plateformes dans deux variables pour pouvoir ensuite modifier leur variable locale pour le bon fonctionnement de notre jeu.

```
95 plat1 = platform(190, 25, 120, 10, WHITE)
96 plat2 = platform(190, 615, 120, 10, WHITE)
```

### e. la mise à jour des images au cours du jeu

Pour pouvoir, effectuer la mise à jour des images lors du jeu, c'est-à-dire dans la boucle principale du jeu : **Game()**. Nous avons créé cette fonction ci-dessous dessinant tout ce qu'il faut dans l'ordre tout en effectuant la mise à jour des images à l'aide des fonctions **draw()** définies précédemment dans nos différentes « **class** », et à l'aide la fonction **pygame.display.update()** de pygame qui sans sa présence ne permet aucune mise à jour.

```
100 def redrawGameWindow():
101     win.fill((0,0,0))
102     win.blit(scoreboard, (500,0))
103
104     plat1.draw(win)
105     plat2.draw(win)
106     PongBall.draw(win)
107
108     # Affiche le score dans le tableau de score
109     displaytext(str(plat1.score), 100, 927, 223, GREEN)
110     displaytext(str(plat2.score), 100, 927, 378, GREEN)
```

```
123     pygame.display.update()
```

#### IV. **Conclusion**

Pour conclure, mon groupe et moi avons aimé cette expérience.

Personnellement, ce projet m'a apporté beaucoup de connaissances en programmation que je continuerai à utiliser dans le futur. Même si notre projet respecte les conditions demandées, j'aurais aimé aller plus loin et ainsi, apporter des améliorations à notre Pong comme par exemple un mode entraînement avec une intelligence artificiel simple qui pourrait prédire la direction de la balle, ou un mode avec vitesse croissante.

#### V. **les différents sites utilisés**

[https://www.w3schools.com/python/python\\_classes.asp](https://www.w3schools.com/python/python_classes.asp)

<https://docs.python.org/3/tutorial/classes.html>

<https://www.youtube.com/>