
PYTHON
&
ALGORITHMIQUE
Bastien Gorissen & Thomas Stassin

Ready ?

Press start!

What We Will Do

war is over

Au menu

Le but d'aujourd'hui est de faire un petit jeu deux joueurs de tir, que nous nommerons sombrement "le jeu des canons".

Le principe est que chaque joueur possède une zone et qu'il peut tirer de cette zone pour essayer d'atteindre la zone de l'adversaire.

Le but de cette exercices est de (re)voir des principes qui vous seront utiles pour le jeu "Astéroïd".

Pas à pas:

Nous utiliserons le module *game_engine* qui est une sorte de couche au-dessus de package *cocos2d*. Vous trouverez ce module ainsi que les assets de bases dans dossier nommer “canons” dans le fichier *zip* où se trouvais aussi les slides ici présents.

Chaque étapes fera l’objet d’une série de slide et sera accompagné de corrections qui se trouvent dans le dossier “corrections” (logique quoi).
Le but est que vous appreniez, donc ne vous ruez pas sur ces fichiers, l’exercice perdrait de son intérêt.

Step 1

Les bases:

Nous allons compléter deux scripts.

le premier se nomme *main.py* et c’est lui que nous “exécuterons” pour jouer au jeu.

Et le deuxième se nomme *game.py* qui lui contiendra les différentes classes de notre jeu.

Les bases:

Dans ce jeu, chaque joueur à une base qui est représenté par un sprite en forme de cercle.

Le joueur 1 aura la base bleu et le joueur 2 aura la base orange. Dans cette étape, nous allons faire en sorte que chacune de ces bases soit placées dans un coin de l'écran. Joueur 1 sera en bas à gauche et joueur 2 en bas à droite.

Le fichier Main.py:

Si vous ouvrez ce fichier vous constaterez qu'il y a déjà des instructions.

```
init((800, 600), "Jeu des canons")
```

la fonction `init` s'occupe de l'initialisation en donnant une dimension (800 x 600) et un nom (Jeu des canons).

```
game = Game()
```

On crée une variable `game` qui servira à accueillir notre jeu, cette variable appartient à la classe `Game` et donc possède une série de méthode dont `run` qui est appéllée ensuite.

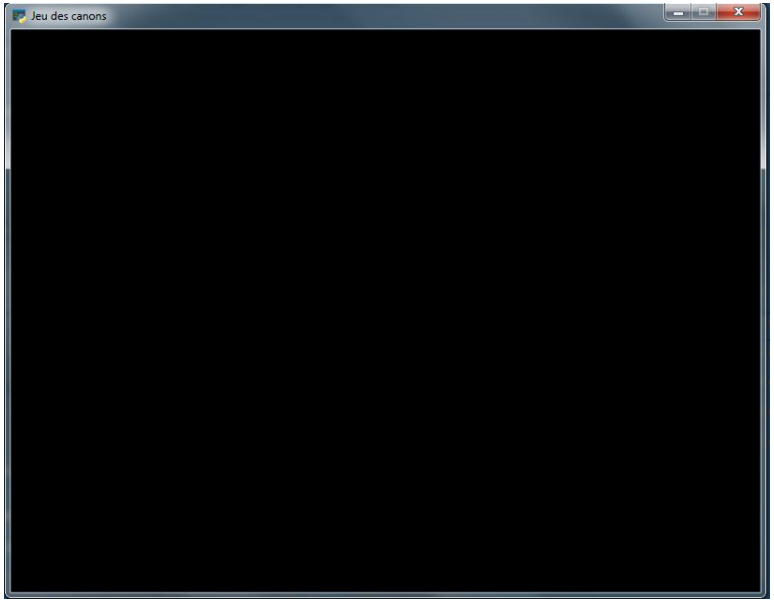
Le fichier Main.py:

```
layer = Layer()

game.add(layer)
```

Un layer est une couche (d'où le nom) où vont s'imprimer vos sprites, nous reviendrons plus tard, mais ce que vous devez retenir c'est que cette couche va accueillir tous les sprite de votre jeu.

Le fichier Main.py:

Si vous lancez le jeu (et donc exécutez le fichier main.py) vous obtiendrez un joli écran noir.



Le fichier Main.py:

Toute le code que nous allons écrire dans ce fichier ce trouvera avant l’appel de la méthode run

Ce qui a du sens, car ce qui serait écrit après ne serait pas pris en compte car le script l'exécuterait après l’exécution du jeu.

Le fichier Game.py:

Ce fichier ne contient pour le moment que des constantes servant à stocker les “assets” qui vont être utilisé par notre jeu.

Notre première étape va être de créer une classe qui hérite de la classe Sprite du module *game_engine.py*

Si on regarde le constructeur de la classe Sprite, il a 5 arguments: `path`, `position`, `scale`, `anchor`, `collision_radius`

Pour aujourd’hui nous ne verrons que `path`, `position` et `anchor`.

Le fichier Game.py:

path: c’est le chemin où le sprite va aller chercher l’image qui le représente.

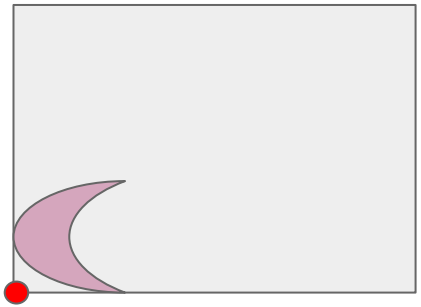
position: représente la position initiale du sprite dans l’ écran

anchor: c’est la position sur le dessin qui le point de référence pour déterminer la position du sprite. On appelle cela l’encre.

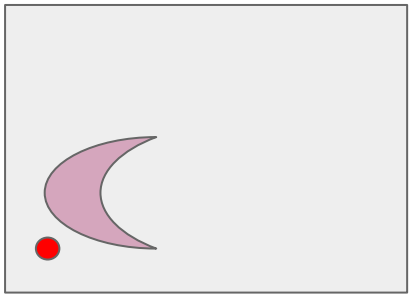
Anchor de l'Encore (euh..):

Prenons un écran de 150 x 10 avec une image de 30 x30

Si je place l'image en 0, 0
avec une encre de 0, 0:



Si je place l'image en 10,10
avec une encre de 0, 0:

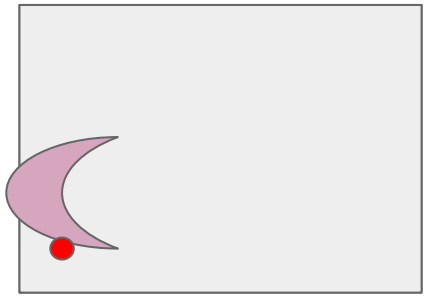


Jusqu'ici rien d'étrange.

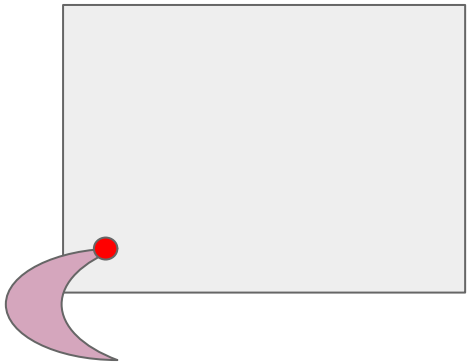
Anchor de l'Encore (euh..):

Mais si je change l'encre

Si je place l'image en 10,10 0 avec une encre de 15, 0:



Si je place l'image en 10,10 avec une encre de 30, 30:



L'encre est la position sur l'image qui sert à déterminer la référence sur laquel va se baser le programme pour déterminer la position sur l'écran.

Les bases des joueurs:

Reprenons.
Il nous faut créer une classe “Base” qui hérite de la classe Sprite.
Cette classe base aura un constructeur qui prendra en paramètre le numéro du joueur (1 ou 2) et sa position initiale sur l’écran.

Les bases des joueurs:

Le constructeur fera appelle au constructeur de la classe parent (à l'aide méthode `super`) en lui donnant le chemin de l'image de la base (contenu dans la variable `BASE_1` ou `BASE_2`).

Il lui passera aussi la position et fixera son encre à `(64, 0)` ce qui correspond au milieu la base de l'image.

Le constructeur mémorisera aussi dans un attribut de la classe le numéro du joueur.

Les bases des joueurs:

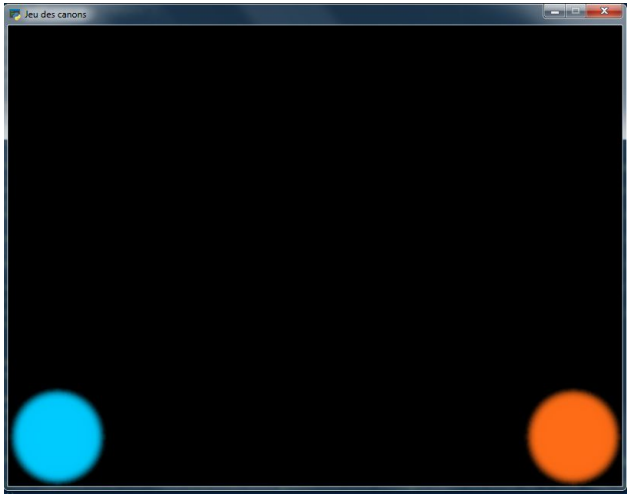
votre classe devrait ressembler à çà:

```
class Base(Sprite):

    def __init__(self, player_number, position):
        self.player_number = player_number
        …

        super().__init__(player_base_path, position, anchor=(64,0))
```

A vous de compléter les “…”.

Finalisation:

Pour finaliser le step 1, dans le fichier main.py, ajouter deux variables (une pour chaque joueur) et ajouter les au *layer*.
Assurez vous que le joueur 1 commence en bas à gauche et le joueur 2 en bas à droite.



Step 2

Evenements :

Afin de capter ce que se passe sur le clavier nous allons apprendre à capter les événements et en particulier les événements lié au clavier.

Il y a deux type d'événements: quand une touche est pressée et lorsqu’elle est relâchée.

Les touches pressées seront captées par la méthode *on_key_press* et les touches relâchées par la méthode *on_key_release.*

Methodes, Evenements ??

Dans une classe héritant de Sprite, si on veut capter les event, il faut définir les méthodes vues précédemment.

si on veut capter la pression de touche, on définira la méthode *on_key_press* de la manière suivante:

def on_key_press(self, key, modifiers):

Il est très important que les arguments `key` et `modifiers` soient présent et dans cette ordre.

Pour *on_key_release* c'est plus ou moins pareil:

def on_key_release(self, key, modifiers):

Methodes, Evenements ??

Que ce soit *on_key_release* ou *on_key_press* il y a deux argument: *key* et *modifiers*. Pour aujourd'hui on ne s'intéressera qu'à *key*.

A chaque fois qu'une touche est pressée les méthodes "on_key_press" de chaque sprite sont appelés.

Et dans l'argument key se trouvera la touche pressée ou du moins le code représentant la touche (qui est un *int*).

Pour aujourd'hui le player 1 utilisera la touche "A" et le player 2 la touche "P" (le code de celles-ci se trouve dans les variables KEY_A et KEY_P).

Methodes et Evenements

Maintenant que vous savez cela vous pouvez écrire un code qui réagit aux évènements du clavier.

Par exemple vous pouvez écrire un code qui vérifie que “A” a été pressé et que le sprite concerné appartient bien au joueur 1 (via l’attribut `player_number`). Et si c’est le cas imprimez (sur la console via `print`) “Player 1 shoot.”

Faites de même pour le joueur 2.

Methodes et Evenements

Normalement si vous pressez les touches “A” et “P” votre console devrait ressembler à ça:

```
player 1 shoot.
player 2 shoot.
```

Il est évident que nous n’allons pas nous contenter d’un *print*.

Step 3

Boulet

Dans cette étape nous allons coder la classe qui va gérer les boulets des canons.

Pour ce faire nous allons devoir parler de MRU (mouvement rectiligne uniforme) et de vecteurs.

On parlera aussi d'accélération et de gravité, mais cela sera pour la prochaine étapes, commençons doucement.

MRU

La distance parcourue par un corps se calcule à l’aide de deux variables:

la vitesse et le temps.

La formule est simple: e = v * t
l’espace parcouru par un corps est égale à la vitesse du corps multiplié par le temps passé à se déplacer.

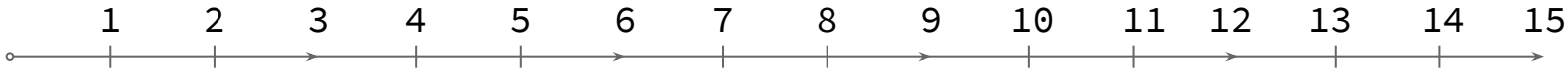
Exemple:

Si vous aller à 3 m/s pendant 5s vous aurez parcouru 15m

MRU

On peut représenter les 3 m/s par une flèche valant 3.

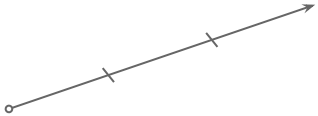
Cela représente le nombre de mètres (3) effectué en une seconde. Et donc si je mets bout à bout 5 fois la même flèche:



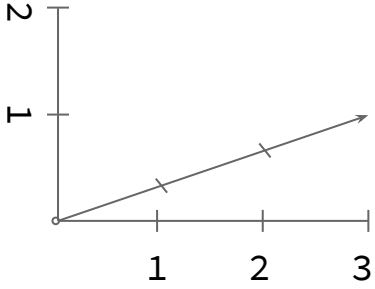
On voit bien que la distance parcourue est de 15 m (pour 5 secondes).

MRU

Au lieu d'aller que sur la droite ma vitesse pourrait aussi monter un peu.

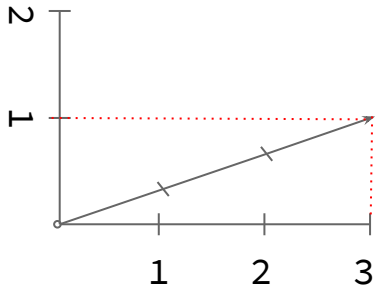


Ce mouvement peut être décomposé en du partie, une partie allant vers la droite et un partie montant



MRU et Vecteur

On peut dire que chaque seconde, on va de 3 vers la droite et de 1 vers le haut.



Et donc on dira que ce mouvement à une vitesse de 3, 1.

Cette représentation est appelée un vecteur.

Voilà, on y est.

Revenons à nos boulet

On va créer une classe Bullet qui héritera aussi de Sprite.

le constructeur prendra le numéro de joueur, la position initiale et la vitesse.

le chemin de l'image utilisée avec le constructeur du parent (`super().__init__`) sera celle stockée dans `BULLET_1` ou `BULLET_2` en fonction de si le numéro du joueur est 1 ou deux. L'encre de l'image sera `8, 8` ce qui correspond au milieu du boulet.

Comme pour la Base, on stockera le numéro du player dans une variable de classe et on fera de même pour la vitesse.

Boulet...

Notre classe Bullet ressemble à ceci:

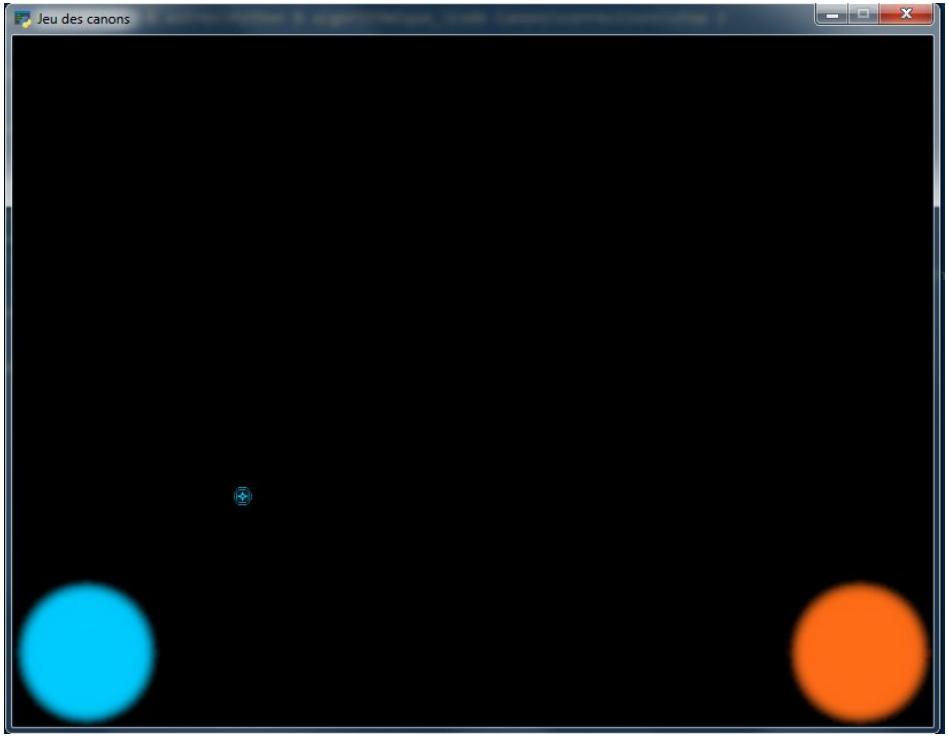
```
class Bullet(Sprite):

    def __init__(self, player_number, position, speed):

        self.player_number = player_number
        self.speed = speed

        if player_number == 1:
            bullet_player_path = BULLET_1
        else:
            bullet_player_path = BULLET_2

        super().__init__(bullet_player_path, position, anchor=(8, 8))
```

Ce qui est, certes fort proche de la classe Base.
Enfin de continuer l'exercice, on va créer une variable bullet dans notre script main.py et l'ajouter dans le layer.
Sa position de départ sera `200,200` et sa vitesse sera de `10, 10` (pour le numéro du joueur je vous laisse choisir).

NB: Cette variable sera retirée dans l'étape prochaine

Boulet...

Normalement votre écran si vous lancez le jeu devrait ressembler à ça:



Du mouvement

Donnons un peu de vie à ce boulet:

Pour le moment même si on a préciser la vitesse, le boulet reste fixe.

Pour mettre à jour la position de boulet en fonction de sa vitesse et du temps passé on devoir définir la méthode update de notre boulet.

update ce défini comme suit:

```
def update(self, dt):
```

Du mouvement

Pour cette méthode on a besoin du comportement de l'ancêtre donc on va faire appelle à `super().update`

Donc la méthode actuellement ressemblera à ça:

```
def update(self, dt):
    super().update(dt)
```

Du mouvement

A chaque “frame” du programme, toutes les méthodes updates de tout les Sprites sont appelée et celles-ci reçoivent dans leur variable `dt` le temps passée depuis la dernière frame (`dt` veut dire *delta time*).

On appelle frame le rafraîchissement entier de l’écran. Et donc dt reçoit le temps qu’il a fallu à l’ordinateur pour tout afficher à l’écran.

Donc on a une vitesse (`self.speed`) et du temps (`dt`) on peut donc calculer un déplacement.

Vecteur et position

On va devoir calculer séparément le déplacement en x et en y à l'aider de la vitesse.
La vitesse en x étant donnée par `self.speed[0]` et celle en y par `self.speed[1]`

le déplacement en x est donc égale à
`self.speed[0] * dt`
Celui en y est `self.speed[1] * dt`

en résumé:

```
dx = self.speed[0] * dt
dy = self.speed[1] * dt
```

Vecteur et position

Il ne reste plus qu'à ajouter ces deux déplacement à la position du boulet (`self.position`) et celle-ci sur mise à jour.

Normalement si vous avez correctement fait votre travail le boulet devrait se mettre à bouger sur l'écran (pas méga vite, j'en conviens.

Step 4