



# Pygame rapport

## Résumé

L'augmentation explosive de la demande de développement de jeux a permis à n'importe qui de créer des jeux. Nous avons créé un jeu à partir de connaissances de base en codage et en théorie simple de la mathématique. Le code utilisé et la théorie mathématique ont mis en œuvre le jeu avec des algorithmes faciles à comprendre

# La production de jeu vidéo avec Pygame avec la théorie de mathématique et la connaissance basique de codage

AHN HARAM

8 janvier 2022

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	scénario . . . . .	4
1.1.1	le processus de production . . . . .	4
1.1.2	le règle général du jeu vidéo à réaliser . . . . .	4
<b>2</b>	<b>Algorithme</b>	<b>5</b>
2.0.1	charger des fichiers de l'image . . . . .	5
2.0.2	harger l'image du fond . . . . .	5
2.0.3	personnage . . . . .	5
2.0.4	charger des poissons et des empêchements . . . . .	5
2.0.5	la création de hameçon et de cordeau . . . . .	5
2.0.6	Le mouvement de hameçon . . . . .	6
2.0.7	Collusion des images . . . . .	7
2.0.8	démonstration mathématique . . . . .	7
2.0.9	convertir vers le code . . . . .	8
2.0.10	La formule de calcul sous la condition que la collusion arrive . . . . .	8
<b>3</b>	<b>conclusion</b>	<b>9</b>

# 1 Introduction

Cela ne fait pas longtemps que le jeu est inclus dans le domaine académique. Le jeu a existé depuis les débuts de la civilisation humaine, Or le jeu n'est pas traité comme un sujet d'étude avant le 20ème siècle. L'origine peut être attribuée à 'Homo Ludens'<sup>1</sup> de Johan Huizinga<sup>2</sup> en 1938. Johan Huizinga s'intéresse au jeu comme une étude initialement. Il définit la racine de l'Homme comme le jeu, et dans ce cas, la conception du jeu contient le plaisir moral et la créativité humaine. Il découvre également que le jeu remonte aux cérémonies du chamanisme de l'époque ancienne. Il contribue à ce que le jeu parvienne à avoir une stature en tant qu'étude légitime.

Le commencement du jeu vidéo électronique est connu comme 'Tennis or Two'. Il crée le jeu pour amuser des visiteurs de laboratoire Brookhaven<sup>3</sup>, ce jeu est non seulement l'origine de jeu commercial mais inspire aussi le jeu 'pong' qui sert à la massification du jeu vidéo électronique. Steve Russell(1962)<sup>4</sup> produit le jeu 'space war' qui est disponible sur ordinateur, et Ralph Henry Baer<sup>5</sup> fabrique le jeu 'Odyssey' qui fonctionne sur télévision. Nolan Bushnell(1972)<sup>6</sup>, conçu comme le père de jeu vidéo électronique, fonde l'entreprise de jeu vidéo Atari<sup>7</sup> et produit le jeu 'Computer Space', 'Pong'. Ses jeux réussissent à populariser le jeu vidéo.

Après que l'industrie du jeu vidéo parvienne à produire d'importants profits commerciaux, les plateformes de jeu indépendantes sont apparues avec la réussite de l'industrie du jeu vidéo. Grâce au développement de plateformes de jeu indépendantes, des personnes appartenant à différents groupes professionnels participent à la fabrication de jeux vidéo. On s'intéresse alors à savoir si n'importe qui peut produire un jeu. Dans ce rapport, nous décrivons qu'il est possible de réaliser le jeu vidéo avec une théorie mathématique simple et un niveau de codage basique. Notre but est de focaliser à réaliser les fonctions fondamentales d'un jeu avec un bas niveau de code.

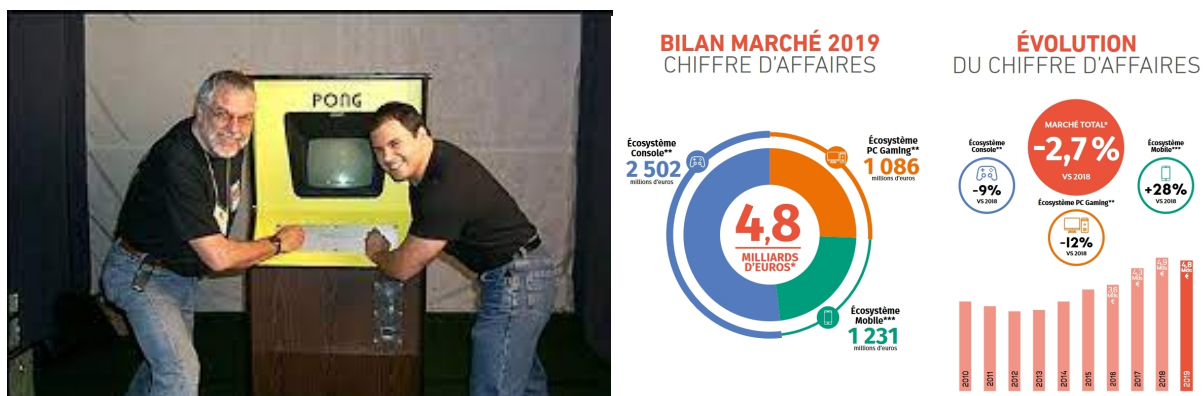


FIGURE 1 – le jeu initial et le marche de jeu vidéo actuelle

1. Homo ludens est une expression utilisée pour la première fois par Johan Huizinga dans son ouvrage Homo ludens, essai sur la fonction sociale du jeu

2. Johan Huizinga : né le 7 décembre 1872 à Groningue et mort le 1er février 1945 à De Steeg, est un historien néerlandais, spécialiste de l'histoire culturelle dans la lignée de Jacob Burckhardt

3. le laboratoire national de Brookhaven ou Brookhaven National Laboratory (BNL) en anglais, est un laboratoire national américain basé à Brookhaven, dans le hameau d'Upton, sur Long Island

4. Steve Russell ( Stephen Russel) dit Slug : né en 1937, est informaticien et un des programmeurs à l'origine du jeu 'Spacewar'

5. Ralph Henry Baer : né le 8 mars 1922 à Pirmasens, et mort le 6 décembre 2014 à Manchester aux États-Unis est un inventeur germano-américain. Baer a apporté de nombreuses contributions au domaine du jeu vidéo, notamment la console à usage domestique raccordée à la télévision. Il est parfois surnommé « le père des jeux vidéo ». La National Medal of Technology and Innovation lui est remise en 2006. Il est introduit au National Inventors Hall of Fame en 2010.

6. Nolan Bushnell : né le 5 février 1943 à Clearfield dans l'Utah, est un pionnier de l'industrie du jeu vidéo aux États-Unis. Il est le concepteur de Pong et le fondateur d'Atari

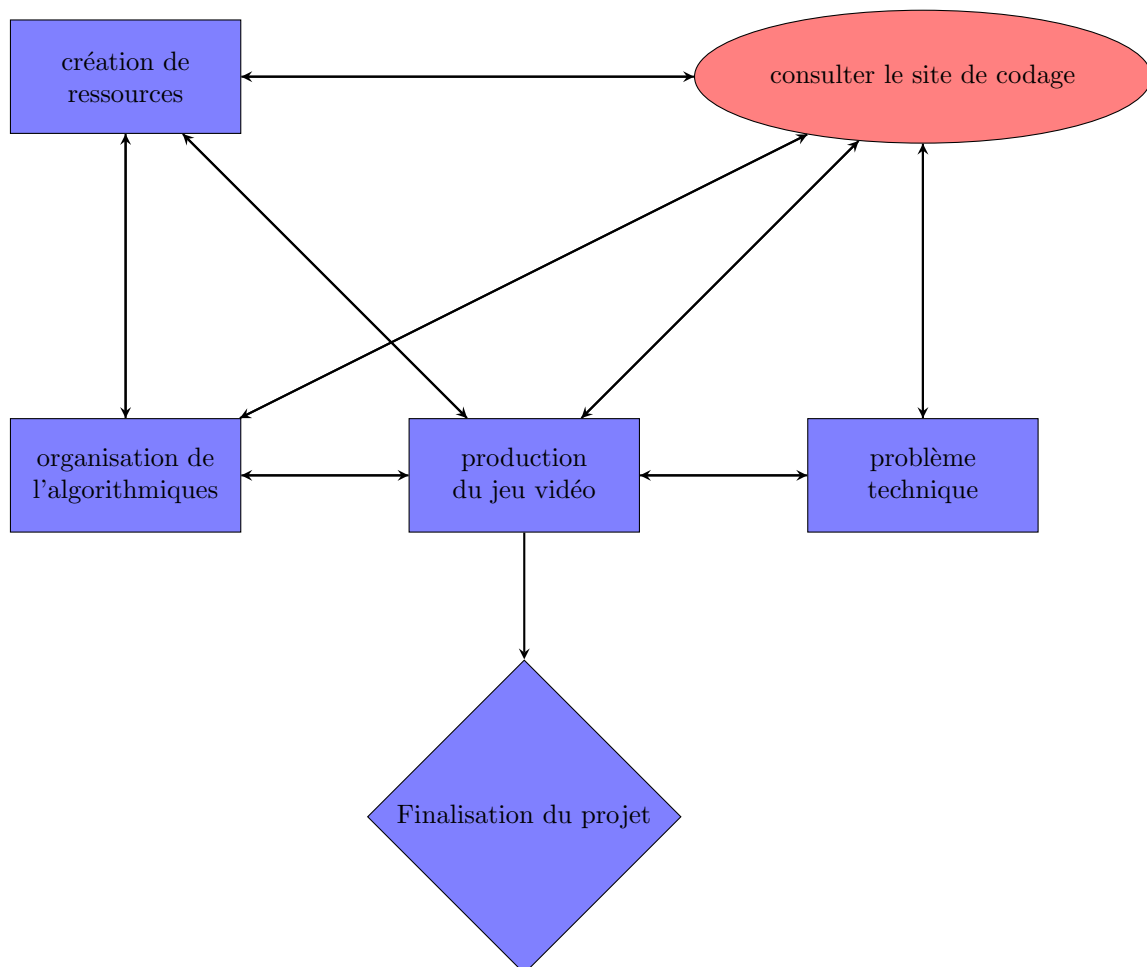
7. Atari est une entreprise française de jeu vidéo, dont le siège est situé à Paris, en France.

## 1.1 scénario

Pour animer ce projet, on fixe le principal basique

1. logique
  - Comme le sujet de ce rapport, notre but est de réaliser le jeu simple avec une connaissance basique de codage, dès lors le jeu à réaliser doit être simple, la logique aussi doit être simple.
2. collusion
  - Pour réaliser la collusion entre un hameçon et un poisson, on a besoin de la théorie de la fonction primaire.
3. la fonction utilisée pour réaliser le jeu
  - Pour accéder aux informations nécessaires à propos de la fonction, nous consultons les sites suivants :
    - Pygame Tutorial
    - Stackoverflow

### 1.1.1 le processus de production



### 1.1.2 le règle général du jeu vidéo à réaliser

Il y a une limite du temps, il faut cliquer quand l'hameçon qui bouge arrive à l'angle où des poissons se situent. Si des utilisateurs pêchent l'empêchement, ils sont pénalisés. S'ils arrivent au note goal, ils gagnent

## 2 Algorithme

### 2.0.1 charger des fichiers de l'image

Tout d'abord, il faut appeler le fichier afin de créer le jeu. Il existe diverses méthodes pour charger le fichier sur Pygame.

### 2.0.2 charger l'image du fond

D'abord, il faut mettre `os.path.dirname(__file__)` dans `current_path` et créer la valeur `background` qui enregistre l'adresse de fichier de l'image du fond. On utilise la fonction `os.path.join` pour retrouver l'adresse du fichier et le nom et appeler l'image avec la fonction `pygame.image.load`. Ensuite, on charge l'image de background avec la fonction `screen.blit`. La valeur pour la fonction `screen.blit` doit être (0,0) et on met à jour avec la fonction `pygame.display.update()` sur l'écran.

```
Background = pygame.image.load(os.path.join(current_path,"background_modif_size.png"))
```

### 2.0.3 personnage

Afin de diversifier le moyen de charger le fichier de l'image, on n'a pas utilisé la fonction `os.path.join`. Il faut créer la valeur `Character` et on enregistre l'adresse du fichier par le chemin définitif dans la fonction `pygame.image.load()`. On utilise la fonction `get.rect().size` pour calculer la longueur et la hauteur de personnage.

```
Character = pygame.image.load(os.path.join(current_path,"personnage.png"))
```

### 2.0.4 charger des poissons et des empêchements

Il est possible de charger des images de poissons et des empêchements de la manière mentionnée dans les parties 2.0.1 et 2.0.2 dans le cas où il y a beaucoup de fichiers qui chargent, cette manière est très inefficace et cause des difficultés pour gérer les fichiers. Afin de résoudre ces problèmes, on crée une liste afin de les gérer facilement et on ajoute une méthode utilisée pour enregistrer des valeurs de hauteurs et des longueurs des fichiers. Il faut hériter `Sprite` offert par le Python dans notre algorithme. Cependant, il est indépendant de créer une classe pour hériter `Sprite`, et on utilise aussi `super._init()` pour la valeur initiale. Il est obligé de créer des variables membres pour user cette fonction pour cela, on crée une valeur de `self.image` dans la classe et on utilise la fonction `get.rect()` pour stocker les informations de la position. On fixe que la valeur de `center` est égale à la valeur de la position.

On utilise une méthode `pygame.sprite.Group()` à la façon de créer un group des informations stocké dans une liste `image_group`, on gère les informations par une méthode `setup_poisson()`. On ajoute les fichiers supplémentaires avec la formule tuple et après on use la fonction de `add`.

```
def setup_poisson() :
    small_poisson = (Poisson(les_images[0],(400,700),poisson_petit_price,poisson
    _petit_vitesse)) image_group.add(small_poisson)
```

Cependant il faut créer deux lignes inutiles, on choisit d'ajouter le fichier dans la liste `Image_group` avec la fonction `add` directement.

```
def setup_poisson():
    image_group.add(Poisson(les_images[0], (400,700),poisson_petit_price,poisson_petit_
    vitesse))
```

### 2.0.5 la création de hameçon et de cordeau

On charge le fichier de l'image de l'hameçon par la même méthode avec 2-2 nous définissons une classe dans notre algorithme pour donner la fonction supplémentaire. On donne la valeur de position (1100,420), cette valeur est identique avec les coordonnées d'une fonction primaire.

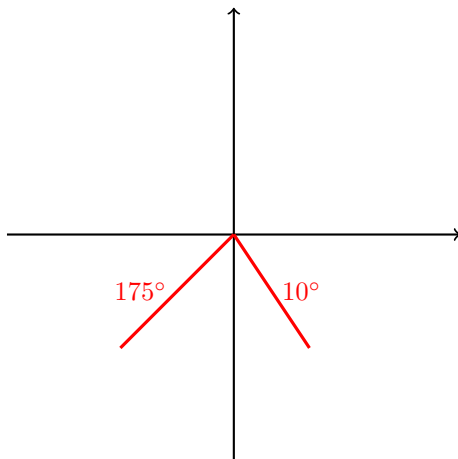
Quand on essaie à mettre à jour en utilisant la fonction de `draw`, on rencontre le message de l'erreur, car chaque sprite ne offre pas la fonction de `draw`. Pour résoudre ce problème, on définit une méthode dans la classe de `Galgori`. Un hameçon doit bouger d'un côté à l'autre, pour cela il faut charger l'image de hameçon dans la position un peu loin du point de centre, alors on définit le constant valeur 40 nommé

`default_offset_x_galgori` avec le formulaire `int` et mettre à jour par la fonction `pygame.math.Vector2`. Afin de fabriquer un cordeau on utilise la fonction `draw.line()` offrit Python.

```
self.offset = pygame.math.Vector2(default_offset_x_galgori,0)
```

### 2.0.6 Le mouvement de hameçon

d'expliquer d'abord, une gamme de mouvements. On fixe  $175^\circ$  pour la gamme maximale de mouvement d'hameçon et  $10^\circ$  pour la gamme initiale de mouvement de hameçon. Afin de réaliser le mouvement, on ajoute une code qui additionne ou soustrait la valeur nommé `self.angle_speed` contenant le valeur `int 2.5` au gamme initiale de mouvement de hameçon dans une méthode `def __init__()` et on ajoute une méthode `def update()` pour calculer la gamme de mouvement.



```
def __init__(self, image, position) :
    super().__init__() :
    self.image = image
    self.rect = image.get_rect(center = position)
    self.original_image = image
    self.direction = LEFT
    self.angle_speed = 2.5
    self.angle = 10
    self.direction = LEFT
    self.angle_speed

def update(self) :
    if self.direction == LEFT : self.angle += self.angle_speed
    elif self.direction == RIGHT :
        self.angle -= self.angle_speed
```

De sorte de réaliser les caractéristiques d'animation avec le code qui calcule la gamme de mouvement du hameçon, on ajoute la fonction `pygame.transform.rotaezoom` dans une méthode nommé `rotate()`. Or, on estime l'image qui bouge hors de la gomme fixé afin de régler ce problème, on refixe la gamme de mouvement par `self.position`.

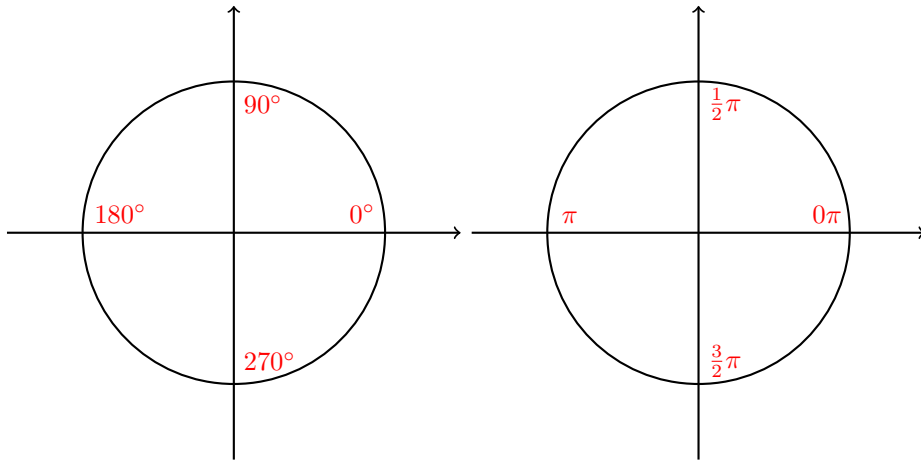
```
def rotate(self) :
    self.image = pygame.transform.rotaezoom(self.original_image,- self.angle, 1)
    self.rect = self.image.get_rect(center = self.position + offset_rotated)
```

### 2.0.7 Collusion des images

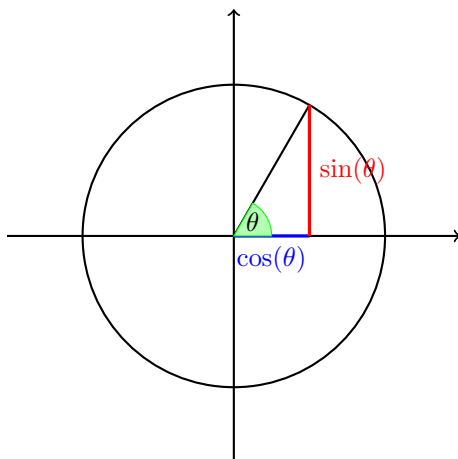
Pour réaliser la collusion entre un hameçon et les poissons, on programme que le poisson est éliminé dans une liste nommé `image_group` si un hameçon atteint les valeurs de coordonnées des poissons. Cependant, il est impossible de réaliser l'animation de pêche réellement, pour cela on utilise la théorie mathématique basique, plus exactement, la fonction trigonométrique.

### 2.0.8 démonstration mathématique

On convertit la valeur de seta vers le radians.



```
import math
print("radians 0°:"
,math.radians(0)," \nradians 90°:",math.radians(90)," \nradians 180°:",math.radians(180),"
 \nradians 270°:",math.radians(270))
Returne :
radians 0° : 0.0
radians 90° : 1.5707963267948966
radians 180° : 3.141592653589793
radians 270° : 4.71238898038469
```





Soit  $r$  est égale à 1

$$\sin \theta = \frac{y}{r}$$

Alors on peut dire  $\sin \theta = y$

$$\cos \theta = \frac{x}{r}$$

Alors on peut dire

$$\cos \theta = x$$

Par conséquent, grâce au principe de la fonction triangulaire, nous pouvons obtenir les coordonnées

### 2.0.9 convertir vers le code

$$\sin \theta = \frac{y}{r}$$

$$\cos \theta = \frac{x}{r}$$

On peut simplifier cette formule.

$$r \times \sin \theta = y$$

$$r \times \cos \theta = x$$

On peut convertir cette formule comme le code suivant dans son algorithme :

```
import math
```

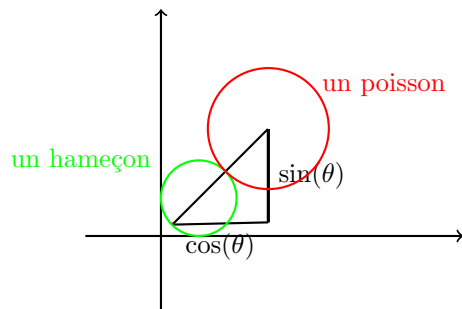
```
r = self.rect.size[0] // 2 \#pie
```

```
rad_angle = math.radians(angle) \#angle
```

```
x = r * math.cos(rad_angle)
```

```
y = r * math.sin(rad_angle)
```

### 2.0.10 La formule de calcul sous la condition que la collision arrive



Soit  $r' + r = L$

$$\sin \theta = \frac{y}{L}$$

$$\cos \theta = \frac{x}{L}$$

On peut dire

$$L \times \sin \theta = y$$

$$L \times \cos \theta = x$$

Alors on peut convertir vers code :

```
self.rect.center = ( position[0] + x, position[1] + y)
```

### 3 conclusion

Nous avons réussi à réaliser un jeu simple avec une logique et une théorie simple de mathématique. Contrairement au plan initial, il y avait des parties difficiles à mettre en œuvre avec des théories mathématiques très élémentaires, mais le jeu a été réalisé au niveau mathématique du lycée. Il n'est pas difficile de réaliser un jeu avec une compréhension mathématique de niveau lycéen et des connaissances basiques de codage.

Or, du fait que la technique de production de jeu vidéo électronique, des performances informatiques progressent de jour en jour, ainsi de nouvelles plateformes sont apparues comme les smartphones, et alors les capacités demandées aux développeurs de jeu vidéo deviennent élevées. En plus, Pour satisfaire aux demandes variées des consommateurs sur le marché du jeu vidéo, le processus de la production de jeu vidéo devient sophistiqué et subdivisé. En conclusion, le niveau plus élevé de codage sera demandé afin de réaliser un jeu actuellement.