

PACMAN

ATELIER DE JEUX VIDÉO



PACMAN

Bienvenue à ce coding club portant sur la création d'un jeux vidéo !

Aujourd'hui, nous allons explorer le fonctionnement d'un jeux, les interactions avec l'utilisateur, les déplacements des sprites, ...

Durant ce coding club, vous allez donc apprendre à manipuler des variables, à utiliser des classes et des fonctions. Tout cela pour pouvoir ouvrir une fenêtre, afficher des images dans cette fenêtre, bouger ces images et utiliser des évènements.

Objectifs

- ✓ Comprendre les concepts de base du développement de jeux vidéo.
- ✓ Apprendre à utiliser Pygame pour la création de jeux 2D en Python.
- ✓ Concevoir et programmer un jeu de plateforme simple inspiré de "Pacman".
- ✓ Explorer la création de niveaux et d'obstacles.
- ✓ Expérimenter avec la synchronisation de la musique et de l'action du jeu.

Développement d'un "Pacman"

Le développement de jeux vidéo est un domaine passionnant et créatif qui allie programmation, conception artistique et narration interactive pour créer des expériences ludiques uniques. Dans cet atelier, nous plongerons dans l'univers du développement de jeux en utilisant le langage de programmation `Python` et la librairie `Pygame`.

Pour vous donner une idée de ce à quoi ressemble un jeu de plateforme stimulant, nous nous inspirerons du célèbre jeu "Pacman".

Ce jeu, connu pour ses graphismes géométriques, sa musique rythmique et ses niveaux remplis de défis, nous servira de modèle pour comprendre les principes de base de la création de jeux vidéo. Au cours de cet atelier, vous découvrirez les étapes de conception, de programmation et de test nécessaires pour donner vie à un jeu de plateforme captivant, tout en développant vos compétences en programmation `Python`. Préparez-vous à plonger dans le monde fascinant du développement de jeux !

Un peu d'histoire sur "Pacman"

Pac-Man, l'un des jeux vidéo les plus emblématiques de tous les temps, a été créé par le développeur japonais Toru Iwatani et publié par Namco en 1980. Ce jeu d'arcade révolutionnaire a rapidement acquis une renommée mondiale grâce à son concept simple mais captivant. Le joueur incarne Pac-Man, une petite créature jaune qui doit traverser un labyrinthe rempli de points et de fantômes. L'objectif est de manger tous les points sans se faire attraper par les fantômes, Blinky, Pinky, Inky et Clyde.

Pac-Man a introduit des éléments de jeu non linéaire, de stratégie et de comportement des ennemis, ce qui en a fait un succès instantané. Il est devenu un phénomène culturel, inspirant des produits dérivés, des dessins animés, des séries télévisées et même une chanson populaire. Pac-Man est devenu un symbole emblématique du monde du jeu vidéo et a laissé une empreinte indélébile dans l'histoire du divertissement électronique. Il est toujours apprécié par les joueurs du monde entier aujourd'hui.

Qu'est ce qu'une librairie graphique ?

Une librairie graphique, également appelée bibliothèque graphique, est un ensemble de fonctions, de modules, de classes et de ressources pré-écrites qui permettent aux développeurs de créer des éléments graphiques, d'afficher des images, de manipuler des graphiques, de gérer des interfaces utilisateur graphiques (GUI) et d'effectuer diverses opérations liées à la visualisation des données.

Voici les principaux concepts associés à une librairie graphique :

- ✓ **Abstraction** : Les librairies graphiques fournissent une abstraction des détails techniques complexes liés aux graphiques et à l'interaction avec le matériel informatique. Elles permettent aux développeurs de se concentrer sur la logique métier de leur application sans avoir à gérer les spécificités de la gestion graphique.
- ✓ **Réutilisation de code** : Les librairies graphiques contiennent des fonctions et des composants réutilisables qui peuvent être utilisés dans différentes parties d'une application ou dans des projets distincts. Cela économise du temps et de l'effort, car les développeurs n'ont pas à réinventer la roue à chaque fois qu'ils ont besoin de fonctionnalités graphiques.
- ✓ **Compatibilité multiplateforme** : Les librairies graphiques sont souvent conçues pour être compatibles avec plusieurs systèmes d'exploitation, ce qui permet aux développeurs de créer des applications qui fonctionnent sur différentes plateformes, telles que Windows, macOS, Linux, iOS, Android, etc.
- ✓ **Performance optimisée** : Les librairies graphiques sont généralement optimisées pour la performance, ce qui signifie qu'elles sont conçues pour gérer efficacement les opérations graphiques, minimiser l'utilisation des ressources système et offrir une expérience utilisateur fluide.
- ✓ **Flexibilité et personnalisation** : Les librairies graphiques offrent souvent des options de personnalisation, permettant aux développeurs de modifier l'apparence et le comportement des éléments graphiques pour répondre aux besoins spécifiques de leur application.
- ✓ **Communauté et documentation** : Les librairies graphiques populaires bénéficient généralement d'une communauté active de développeurs, de forums de support, de documentation complète et de tutoriels qui facilitent l'apprentissage et la résolution de problèmes.

Exemples courants de librairies graphiques dans différents langages de programmation :

Python : Pygame, Tkinter, PyQt, wxPython.

C# : WPF (Windows Presentation Foundation), Unity (pour le développement de jeux).

C++ : Qt, SDL, OpenGL (pour la programmation graphique bas-niveau).

En résumé, une librairie graphique est un outil puissant qui simplifie la création d'applications avec des composants visuels, en offrant des fonctionnalités prêtes à l'emploi, des abstractions et une compatibilité multiplateforme, tout en permettant aux développeurs de se concentrer sur la création de leurs applications plutôt que sur les détails techniques des graphiques.

Création du projet de jeu et de la fenêtre de jeu de base

Création d'une fenêtre de jeu :

À cette étape, nous allons mettre en place l'environnement de base pour notre jeu "Pacman" en utilisant Pygame. Nous allons créer une fenêtre de jeu où le jeu se déroulera.

Création d'un projet de jeu :

1. Ouvrez votre environnement de développement Python (comme IDLE, Visual Studio Code, ou un autre que vous préférez).
2. Créez un nouveau dossier pour votre projet de jeu "Pacman" et donnez-lui un nom significatif.
3. Au sein de ce dossier, créez un nouveau fichier Python que vous nommerez "**main.py**". C'est ici que nous écrirons le code de notre jeu.

Importez la librairie Pygame :

```
Terminal
Cobra> cat main.py
import pygame
```

```
Terminal
Cobra> python3 main.py
pygame 2.5.1 (SDL 2.28.2, Python 3.11.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Initialisation de Pygame :

1. Créez une fonction `main()` qui contiendra le code principal de notre jeu. Ajoutez la ligne suivante au début de votre fichier `main.py` :
2. Pour utiliser Pygame, nous devons l'initialiser en appelant la fonction `pygame.init()`. Cela permet à Pygame de préparer les modules nécessaires à l'exécution de notre jeu. Ajoutez la ligne suivante à la fonction `main()` :

```
Terminal
Cobra> cat main.py
import pygame

def main():
    pygame.init()
```

3. Créez deux variable qui contient les dimensions de la fenêtre avec pour valeur en hauteur 600 et en largeur 800.
Utilisez la fonction `pygame.display.set_mode()` pour créer une fenêtre de jeu avec les dimensions que vous avez choisi.
Cette fonction prend en paramètre un tuple contenant les dimensions de la fenêtre. Puis utilisez la fonction `pygame.display.set_caption()` pour donner un titre à votre fenêtre de jeu.
4. Créez une fonction `process_event()` pour la gestion des événements. Cette fonction sera appelée à chaque fois qu'un événement se produira.
Pour ce faire, utilisez la fonction `pygame.event.get()` qui retourne une liste des événements qui se sont produits depuis la dernière fois que cette fonction a été appelée.
Puis utilisez une boucle `for` pour parcourir la liste des événements et détectez les événements de fermeture de la fenêtre et de pression sur la touche **Echap** pour pouvoir quitter le jeu.

```
Terminal
Cobra> cat main.py
import pygame

def process_event():
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                exit()
```

5. Pour afficher la fenêtre de jeu, utilisez la fonction `pygame.display.flip()` qui met à jour l'écran avec les changements que vous avez apportés.

```
Terminal
Cobra> cat main.py
import pygame

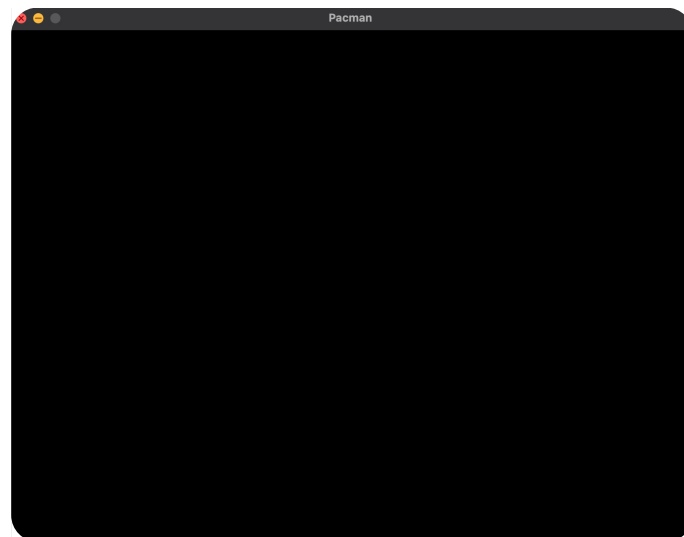
def process_event():
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                exit()

def main():
    pygame.init()
    screen = pygame.display.set_mode((800, 600))
    pygame.display.set_caption("Pacman")

    while True:
        process_event()

        pygame.display.flip()

if __name__ == "__main__":
    main()
```



Création de la map :

Définition de la map

1. Créez une fonction `get_map()` pour définir la carte du jeu en utilisant une matrice (tableau 2D) pour représenter les murs, les pac-gommes, etc. Définissez une variable `map` de type tableau 2D qui contient les valeurs suivantes :

- ✓ 0 : représente un espace vide.
- ✓ 1 : représente un mur horizontal.
- ✓ 2 : représente un mur vertical.
- ✓ 3 : représente une intersection.



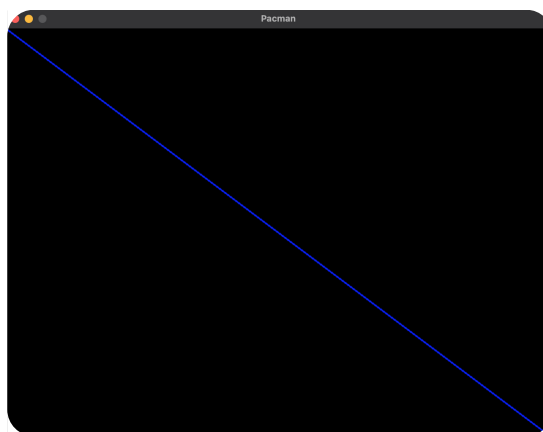
Voici un exemple de définition d'une map.

```
map = [[0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0],
        [0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0],
        [1,3,1,1,1,1,1,1,1,3,1,1,1,1,1,1,1,3,1,1,1,1,1,3,1],
        .....
        [0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0]]
```

Affichage de la map

1. Apprenez à utiliser la fonction `pygame.draw.line()` pour dessiner des lignes dans la fenêtre de jeu. Cette fonction prend en paramètre la variable `screen` pour dessiner dans la fenêtre, la couleur de la ligne et les coordonnées `x` et `y` de départ et d'arrivée de la ligne.

```
pygame.draw.line(screen, (0, 0, 255), (0, 0), (800, 600), 3)
```



Lorsque vous avez terminé, votre fonction `main()` devrait ressembler à ceci :



```
def main():
    pygame.init()
    screen = pygame.display.set_mode((800, 600))
    pygame.display.set_caption("Pacman")

    while True:
        process_event()
        pygame.draw.line(screen, (0, 0, 255), (0, 0), (800, 600))
        pygame.display.flip()
```

2. Créez une fonction `draw_map()` qui prend en paramètre la variable `screen` pour afficher la carte du jeu. Pour ce faire, utilisez une boucle `for` imbriquée pour parcourir la `map` et dessinez les murs en utilisant la fonction `pygame.draw.line()`.

La fonction `enumerate()` permet de parcourir une liste et de retourner à chaque itération un tuple contenant l'index de l'élément et l'élément lui-même.



```
def draw_map(screen):
    map = get_map()

    for i, row in enumerate(map):
        for j, item in enumerate(row):
            if item == 1:
                draw_wall_horizontal(screen, i, j)
            elif item == 2:
                draw_wall_vertically(screen, i, j)
```

3. Créez deux fonctions `draw_wall_horizontal()` et `draw_wall_vertically()` qui prennent en paramètre la variable `screen` et les coordonnées `x` et `y` pour dessiner les murs horizontaux et verticaux. Vous pouvez créer une variable `SIZE` pour stocker la taille d'un bloc.

✓ 32 : représente la taille d'un bloc.



```
def draw_wall_horizontal(screen, x, y):
    pygame.draw.line(screen, (0, 0, 255), (x * 32, y * 32), (x * 32 + 32, y * 32),
                     3)

def draw_wall_vertically(screen, x, y):
    pygame.draw.line(screen, (0, 0, 255), (x * 32, y * 32), (x * 32, y * 32 + 32),
                     3)
```

4. Appelez les fonctions `draw_wall_horizontal()` et `draw_wall_vertically()` pour dessiner les murs horizontaux et verticaux dans la fonction `draw_map()`.



5. Modifier la fonction `draw_map()` pour afficher les intersections.



Ajout des pac-gommes

Création des pac-gommes

1. Créez une fonction `create_dot`, puis utilisez la classe suivante pour créer des Ellipses. Vous devez créer un groupe de sprite pour pouvoir afficher les pac-gommes dans la fenêtre de jeu.
Cette fonction devra parcourir la `map` et créer une pac-gomme à chaque fois qu'elle rencontre une valeur différente de `0`.
Appelez cette fonction dans la fonction `main()` avant la boucle `while` et stocker la valeur de retour dans une variable `dots_group`.



Pour créer un groupe de sprite, utilisez la fonction `pygame.sprite.Group()`.

```
dots_group = pygame.sprite.Group()
```



Voici une implémentation de la classe `Ellipse` que vous pouvez utiliser pour créer des pac-gommes :

```
class Ellipse(pygame.sprite.Sprite):
    def __init__(self, x, y, color, width, height):
        # Call the parent class (Sprite) constructor
        pygame.sprite.Sprite.__init__(self)

        # Set the background color and set it to be transparent
        self.image = pygame.Surface([width, height])
        self.image.fill((0, 0, 0))
        self.image.set_colorkey((0, 0, 0))

        # Draw the ellipse
        pygame.draw.ellipse(self.image, color, [0, 0, width, height])
        self.rect = self.image.get_rect()
        self.rect.topleft = (x, y)
```



Voici un exemple d'implémentation de la fonction.

- ✓ 12 : permet de centrer la pac-gomme dans le bloc.
- ✓ 8 : représente la taille de la pac-gomme.

```
def create_dot():
    map = ...
    dots_group = pygame.sprite.Group()
    for i, row in enumerate(map):
        for j, item in enumerate(row):
            if item != 0:
                dots_group.add(Ellipse(j * SIZE + 12, i * SIZE + 12, WHITE, 8, 8))
    return dots_group
```

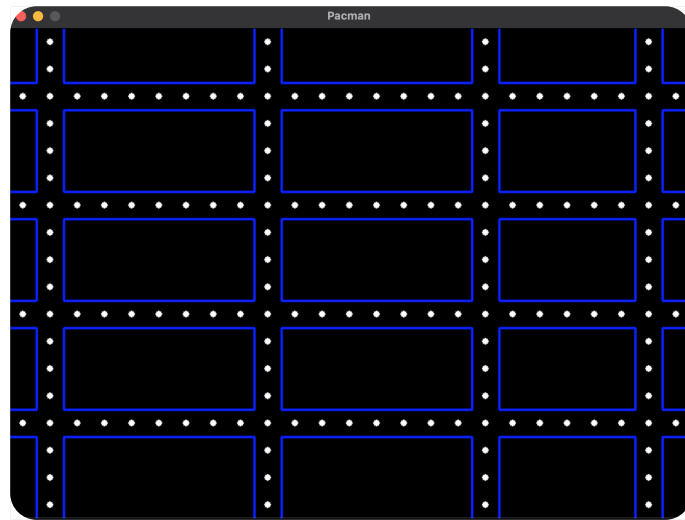
Affichage des pac-gommes

1. Affichez les pac-gommes en utilisant la variable `dots_group` dans la fonction `main()`.



```
def main():
    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Pacman")

    dots_group = create_dot()
    while True:
        process_event()
        draw_map(screen)
        dots_group.draw(screen)
        pygame.display.flip()
```



Ajout de Pacman

Création de Pacman

1. Créez une fonction `load_pacman()` qui prend en paramètre la variable `filename` pour créer un personnage. Cette fonction devra charger en mémoire l'image du personnage et la retourner.
L'image chargée en mémoire devra être assignée à un objet de type `Sprite` qui est une classe de Pygame.
Vous devez avoir une image de Pacman dans le dossier `img` nommée `pacman.png`.



Un objet de type `Sprite` est un objet qui peut être affiché dans la fenêtre de jeu. Il peut être animé, déplacé, etc.

Voici un exemple de la fonction `load_pacman()` qui charge l'image du personnage :

```
def load_pacman(filename):  
    pacman = pygame.sprite.Sprite()  
    pacman.image = pygame.image.load(filename).convert()  
    pacman.image.set_colorkey(BLACK)  
    pacman.rect = pacman.image.get_rect()  
    pacman.rect.topleft = (32, 128)  
    return pacman
```

2. Affichez le personnage dans la fenêtre de jeu en utilisant la fonction `screen.blit()` qui prend en paramètre la variable `screen` pour afficher le personnage, la variable `pacman` qui contient l'image du personnage et les coordonnées `x` et `y` pour positionner le personnage dans la

fenêtre de jeu.



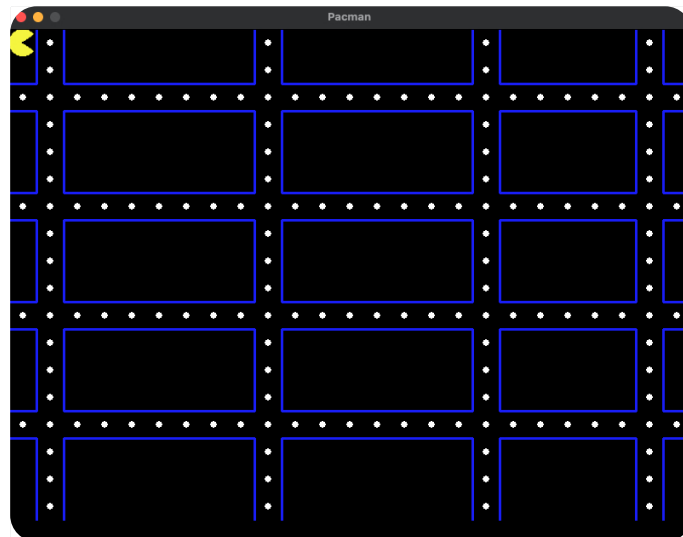
Voici un exemple de la fonction `main()` qui affiche le personnage dans la fenêtre de jeu :

```
def main():
    ...
    pacman = pygame.sprite.Sprite()
    pacman.image = load_pacman("img/pacman.png")
    pacman.rect = pacman.image.get_rect()
    pacman.rect.topleft = (0, 0)
    while True:
        ...

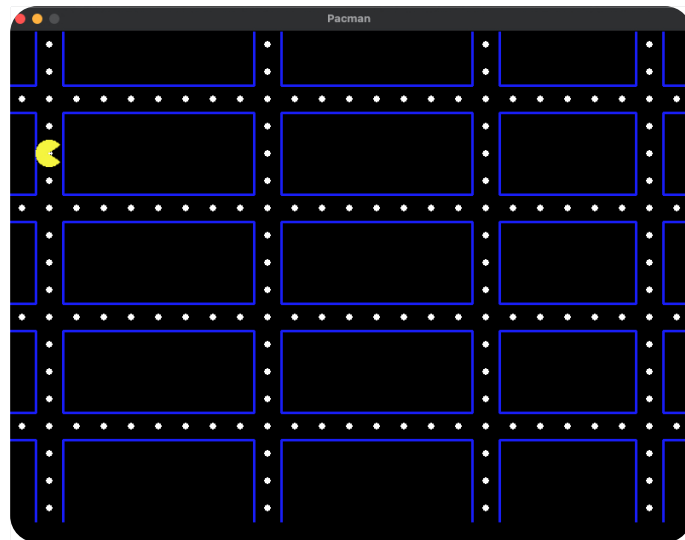
        screen.blit(pacman, pacman.rect)

    pygame.display.flip()
```

Vous devriez voir le personnage dans la fenêtre de jeu en haut à gauche. Exemple :



3. Modifiez les coordonnées `x` et `y` pour positionner le personnage au niveau d'un chemin, comme ci-dessous.



4. Créez une fonction `move_pacman()` qui prend en paramètre la variable `speed` qui contient la vitesse du personnage et la direction du personnage. Cette fonction attribuer la vitesse du personnage dans la direction donnée au vecteur de déplacement du personnage. Mettez à jour les coordonnées `x` et `y` du rectangle du sprite de pacman dans la boucle de jeu.



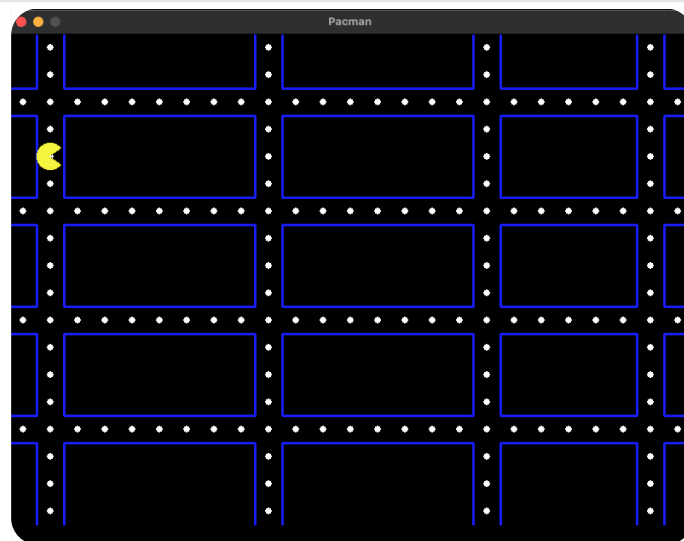
Utilisez la fonction `Vector2` de Pygame pour générer un vecteur de déplacement du personnage. Ajoutez une clock pour gérer le framerate du jeu. Pour cela, utilisez la fonction `pygame.time.Clock()` qui retourne un objet `clock` qui permet de gérer le framerate du jeu. Puis utilisez la fonction `tick()` pour mettre à jour le framerate du jeu.

```
speed = pygame.math.Vector2(0, 0)
```



Voici un exemple de la fonction `process_event()` qui gère les événements de déplacement du personnage :

```
def process_event(speed):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_ESCAPE:
                pygame.quit()
                exit()
            elif event.key == pygame.K_LEFT:
                move_pacman(speed, ...)
            ...
            ...
            elif event.key == pygame.K_DOWN:
                move_pacman(speed, ...)
    elif event.type == pygame.KEYUP:
        if event.key == pygame.K_LEFT:
            stop_move_pacman(speed, "left")
        ...
        ...
        stop_move_pacman(speed, "up")
    elif event.key == pygame.K_DOWN:
        stop_move_pacman(speed, "down")
```



Gestion des collisions

1. Afin de faciliter la gestion des collisions, nous allons créer des sprites qui vont nous permettre de savoir si pacman est en dehors du chemin ou non.
- ✓ Créez une fonction `create_horizontal_blocks()` et une fonction `create_vertical_blocks` pour créer des sprites qui vont nous permettre de détecter les collisions entre le personnage et les murs horizontaux et verticaux.
 - ✓ Modifiez la couleur des blocks pour les rendre invisible, mettez les en noir au lieu de rouge.



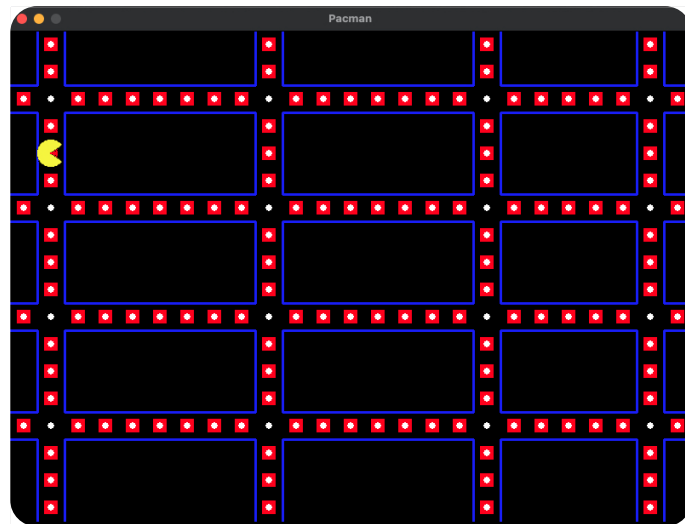
Voici un exemple pour la création d'un groupe de Sprite pour les chemin horizontaux :

```
def create_blocks(x, y, color, width, height):
    ...
    sprite.image = pygame.Surface([width, height])
    sprite.image.fill(color)
    sprite.rect = sprite.image.get_rect()
    sprite.rect.topleft = (... , ...)
    return sprite

def create_horizontal_blocks():
    map = get_map()
    horizontal_blocks = pygame.sprite.Group()
    for i, row in enumerate(map):
        for j, item in enumerate(row):
            if item == 1:
                sprite = create_blocks(j * SIZE + 8, i * SIZE + 8, (255, 0, 0), 16,
                                         16)
                horizontal_blocks.add(sprite)
    return horizontal_blocks
```

2. Affichez les blocks dans la fenêtre de jeu en utilisant la fonction `draw()` dans la boucle de jeu.

Vous devriez obtenir le résultat suivant :



3. Créez une fonction `is_outside(pacman, horizontal_blocks, vertical_blocks)` qui prend en paramètre la variable `pacman` qui contient l'image du personnage et les groupes de sprite verticaux et horizontaux. Cette fonction devra retourner `True` si le personnage est en dehors du chemin et `False` sinon.



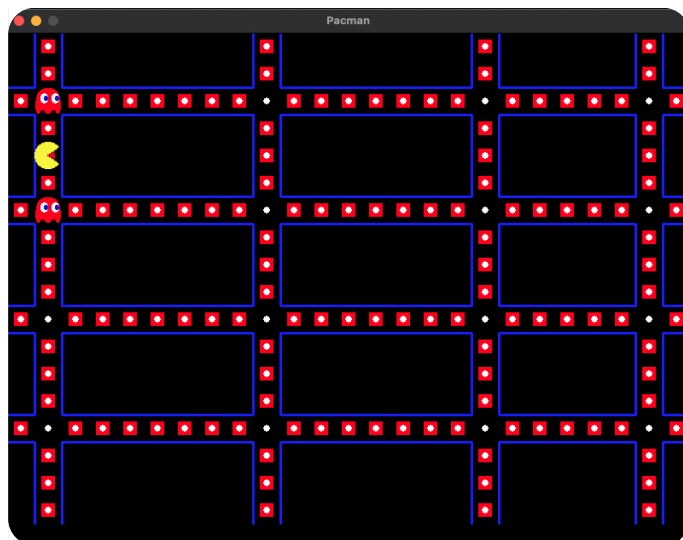
Pour détecter les collisions entre le personnage et les murs horizontaux et verticaux, utilisez la fonction `pygame.sprite.spritecollide()` qui prend en paramètre le personnage, le groupe de sprite et un booléen pour savoir si on veut supprimer le sprite lors d'une collision.

Gestion d'un score

1. Créez une variable `score`, cette variable sera incrémentée à chaque fois que Pacman aura réussi à manger une pac-gomme.
2. Créez une fonction `is_eating(dots_group, pacman)` qui prend en paramètre la variable `pacman` qui contient l'image du personnage et le groupe de sprite des pac-gommes. Cette fonction devra retourner `True` si le personnage mange une pac-gomme et `False` sinon. Appelez cette fonction dans la boucle de jeu, puis incrémentez la variable `score` si le personnage mange une pac-gomme puis affichez le score sur le terminal.

Ajout d'un ennemi

1. Créez une fonction `load_ghost(filename, position)` qui prend en paramètre la variable `filename` pour créer un ennemi et sa position initiale. Cette fonction devra charger en mémoire l'image de l'ennemi et la retourner comme pour le personnage.
2. Créez un groupe de sprite `pygame.sprite.Group()` pour stocker les ennemis, comme pour les pac-gommes, puis affichez les ennemis dans la fenêtre de jeu en utilisant la fonction `draw()`.



3. Créez une fonction `move_ghost(slimes, speed)` qui prend en paramètre le groupe de sprite des ennemis et la vitesse de déplacement des ennemis. Cette fonction devra être utilisé au sein de la boucle de jeu.

Vous devez parcourir le groupe de sprite des ennemis et déplacer chaque ennemi dans la direction opposée à celle du personnage, n'oubliez pas de vérifier si l'ennemi est en dehors de la fenêtre de jeu.

3. Détectez les collisions entre le personnage et l'ennemi en utilisant la fonction `pygame.sprite.spritecollide()` qui prend en paramètre le personnage, le groupe de sprite et un booléen pour savoir si on veut supprimer le sprite lors d'une collision. Si le personnage entre en collision avec l'ennemi, affichez un message de fin de jeu.

Mouvement aléatoire des ennemis

1. Ajoutez des ennemis dans la fenêtre de jeu en utilisant la fonction `load_ghost()` et ajouter l'ennemi dans le groupe de sprite des ennemis.
2. Détectez les intersections et faites changer de direction les ennemis de manière aléatoire. Pour ce faire, créez une fonction `get_intersections()` qui retournera un tableau contenant les positions des intersections. Appelez cette fonction dans la fonction `move_slime()`. Si l'ennemi est sur une intersection, faites changer de direction l'ennemi de manière aléatoire, vous pouvez utiliser la fonction `random.choice()` qui prend en paramètre un tableau et retourne une valeur aléatoire du tableau.



Voici un exemple de la fonction `check_intersection()` qui retourne un tableau contenant les positions des intersections :

```
def get_intersections():
    items = []
    map = ...
    for i,row in enumerate(map):
        for j,item in enumerate(row):
            if item == 3:
                items.append((j * SIZE, i * SIZE))
    return items

def move_slime(slimes, speed):
    for sprite in slimes.sprites():
        sprite.rect.x += speed.x
        sprite.rect.y += speed.y

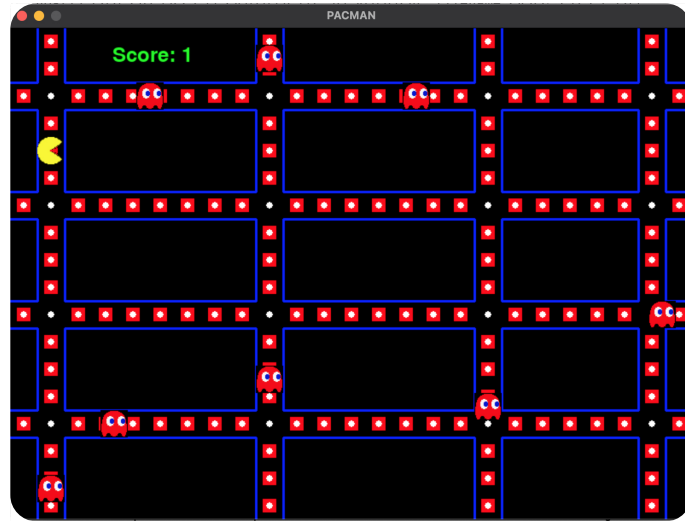
        # Ici vous pouvez verifier si la position du slime n est pas hors de la
        # fenetre de jeu

    if sprite.rect.topleft in get_intersection():
        direction = random.choice(("left", "right", "up", "down"))
        if direction == "left" and speed.x == 0:
            speed.x = -speed.x
            speed.y = 0
        elif direction == "right" and speed.x == 0:
            speed.x = speed.x
            speed.y = 0
        elif direction == "up" and speed.y == 0:
            speed.x = 0
            speed.y = -speed.y
        elif direction == "down" and speed.y == 0:
            speed.x = 0
            speed.y = speed.y
```



Voici un exemple de la fonction `random.choice()` qui retourne une valeur aléatoire du tableau :

```
direction = random.choice(("left", "right", "up", "down"))
```



Allez plus loin

Nous vous invitons à continuer le développement de ce jeu en ajoutant des fonctionnalités et expérimentant avec les concepts que vous avez appris. Voici quelques idées pour vous aider à démarrer :

Ajout des animations de Pacman

Vous trouverez un fichier nommé `walk.png` dans le dossier `img` qui contient les images de Pacman. Utilisez ces images pour créer une animation de Pacman en utilisant la fonction `pygame.time.get_ticks()` pour calculer le temps écoulé depuis le début du jeu.

Vous pouvez trouver des tutoriels sur l'animation de sprites avec Pygame ici :
- [geeksforgeeks](#)

Ajout d'un niveau

Vous pouvez réaliser un changement de niveau une fois que l'utilisateur a mangé toutes les pac-gommes. Pour ce faire, vous devez créer une nouvelle map et réinitialiser les positions du personnage et des ennemis.

Ajout d'un son

Vous pouvez ajouter des sons pour les collisions, les pac-gommes, les ennemis, etc. Pour ce faire, vous devez charger les sons en mémoire et les jouer au bon moment.



Vous trouverez au sein de la [documentation](#) de Pygame les informations nécessaires pour ajouter des sons dans votre jeu.

Affichez le score

Vous pouvez afficher le score dans la fenêtre de jeu en utilisant la fonction `pygame.font.SysFont()` pour charger une police de caractère et la fonction `render()` pour afficher le score.

Ajout d'un menu

Si vous souhaitez ajouter un menu avant de démarrer une partie, vous pouvez utiliser des bibliothèques tels que :

- Pygame-menu.
- Tkinter.

Programmation orientée objet

Vous pouvez réécrire le code de ce jeu en utilisant la programmation orientée objet. Pour ce faire, vous devez créer des classes pour les différents éléments du jeu (personnage, ennemis, pac-gommes, etc.) et les méthodes associées.



Vous trouverez au sein de la [documentation de Python](#) les informations nécessaires pour créer des classes et des méthodes.

Annexes

Bases du langage python

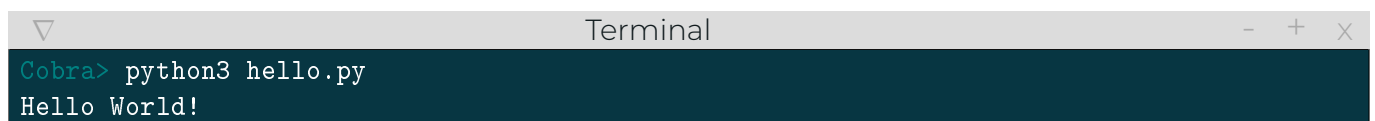
Python est un langage de programmation de haut niveau réputé pour sa syntaxe claire et lisible. Les blocs de code sont délimités par l'indentation, ce qui signifie que l'espacement correct est essentiel. Les déclarations se terminent généralement par des sauts de ligne, mais vous pouvez également utiliser un point-virgule pour les séparer.

Hello World :

La première chose à faire lorsque vous apprenez un nouveau langage de programmation est d'écrire un programme qui affiche "Hello World". En Python, vous pouvez le faire en une seule ligne :

```
print("Hello World!")
```

Pour exécuter ce programme, vous pouvez utiliser l'interpréteur Python en ligne de commande :

A terminal window titled "Terminal" with standard window controls (minimize, maximize, close). The prompt is "Cobra>". The command entered is "python3 hello.py". The output displayed is "Hello World!".

```
Cobra> python3 hello.py
Hello World!
```

Variables et Types de Données :

En Python, vous pouvez créer des variables pour stocker différentes sortes de données, comme des nombres, des chaînes de caractères, des listes, des dictionnaires, etc. Les noms de variables sont sensibles à la casse et doivent commencer par une lettre ou un soulignement (_). Exemples de types de données :

```
nombre_entier = 42
nombre_decimal = 3.14
chaine = "Bonjour, Python!"
liste = [1, 2, 3, 4]
tableau_2d = [[1, 2], [3, 4]]
dictionnaire = {'cle': 'valeur'}
```


Opérations de Base :

Python prend en charge les opérations mathématiques de base (+, -, *, /) ainsi que les opérations logiques (and, or, not). Exemples :

```
somme = 5 + 3
difference = 10 - 2
produit = 4 * 6
quotient = 12 / 3

vrai_et_faux = True and False
vrai_ou_faux = True or False
non_vrai = not True
```

Structures de Contrôle :

Python propose des structures de contrôle telles que les boucles (for, while) et les instructions conditionnelles (if, elif, else) pour gérer le flux d'exécution du programme.

```
for i in range(5):
    print(i) # Affiche les nombres de 0 a 4

x = 0
while x < 5:
    print(x)
    x += 1 # Incrémente x a chaque iteration

if nombre > 10:
    print("Nombre supérieur a 10")
elif nombre == 10:
    print("Nombre égal a 10")
else:
    print("Nombre inférieur a 10")
```

Fonctions :

Les fonctions sont des blocs de code réutilisables qui effectuent une tâche spécifique. Elles peuvent prendre des paramètres en entrée et renvoyer une valeur en sortie.

```
def carre(x):
    return x * x
```

Valeur de Retour :

Les fonctions peuvent renvoyer une valeur en sortie. Si aucune valeur n'est renvoyée, la fonction renvoie None.

```
def carre(x):  
    return x * x  
  
print(carre(5)) # Affiche 25
```

Listes et boucles :

Les listes sont des collections ordonnées d'éléments. Elles peuvent contenir des éléments de différents types, et vous pouvez ajouter ou supprimer des éléments à tout moment. Les boucles for sont souvent utilisées pour parcourir les listes.

```
nombre = [1, 2, 3, 4, 5]  
for nombre in nombres:  
    print(nombre)
```

Classe et méthode :

Les classes permettent d'architecturer un programme informatique. Elles permettent de représenter des entités qui sont réutilisable facilement et de fournir des fonctionnalités au travers de méthodes.

```
class Person(name, age):  
    def __init__(self):  
        super().__init__()  
        self.name = name  
        self.age = age  
  
    def display():  
        print("Bonjour je suis self.name{self.age}")
```



DIVERSITY
by **EPITECH**