CODING CLUB

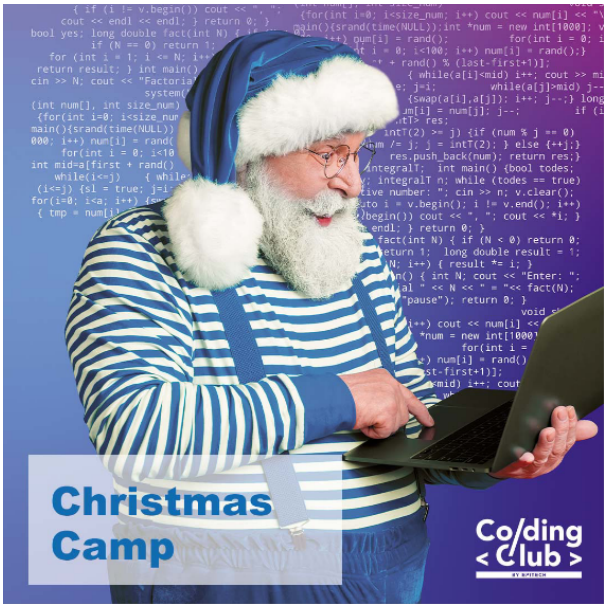
EPITECH LYON

Free Santa

VERSION 1.0.0

DIVERSITY by EPITECH

Free Santa



INTRODUCTION

Le Père Noël part comme tous les ans faire sa tournée, quand soudain un grapin s'accroche à son précieux traîneau alors qu'il survole une île nordique. Trainé au sol, il se retrouve rapidement en tête à tête avec le parrain de la pègre locale tandis que son véhicule est confisqué. Il n'a plus qu'une solution pour pouvoir reprendre sa tournée : aider ses ravisseurs à résoudre des énigmes et sortir de ce labyrinthe infernal afin d'avoir eux aussi un Noël agréable ! Pour cela nous allons créer grâce à python et pygame un jeu de labyrinthe qui va permettre au père noël d'être libéré. pygame est une librairie assez basique mais qui est très simple à prendre en main. De plus elle est très bien documentée et il y a beaucoup de tutoriels sur internet. Vous avez ici le lien vers la documentation de la librairie : https:// www.pygame.org/ docs/.

INSTALLATION

Pour pouvoir faire ce projet nous avons deux solutions possibles: - Installer python et pygame sur votre machine - Utiliser un environnement de développement en ligne

Si vous voulez faire le premier choix, libre à vous et on vous aidera à l'installer mais pour des raisons de simplicité nous allons utiliser un environnement de développement en ligne. Pour cela il vous suffit de vous rendre sur https://repl.it/languages/pygame et de vous créer un

compte. Une fois que vous avez créé votre compte vous pouvez créer un nouveau projet et vous aurez un environnement de développement python en ligne. Vous pouvez maintenant commencer à coder.

LE CODE

Explication du code

Une fois que vous avez créé votre projet, vous devez être sur une interface avec un fichier main.py contenant du code ressemblant à ceci:

```
import pygame, sys

pygame.init() # Initialise pygame
screen = pygame.display.set_mode((400, 300)) # Définir les dimensions de la fe
pygame.display.set_caption('Hello World!') # Définir le titre de la fenètre
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```

`import pygame, sys`

Cette ligne permet d'importer les librairies dont on aura besoin. La librairie pygame permet de faire des jeux en python et la librairie sys permet de faire des opérations sur le système.

Vous avez ensuite la boucle de jeu:

```
while True:
    clock.tick(60) # Ajoutez cette ligne
    for event in pygame.event.get():
        if event. type == QUIT:
            pygame.quit()
            sys.exit()
        pygame.display.update()
```

La boucle while True permet d'éxécuter notre code sans s'arrêter. les 3 lignes suivantes:

```
for event in pygame.event.get():
    if event.type == QUIT:
```

```
            pygame.quit()
            sys.exit()
```

Servent à détecter un évenement, dans ce cas on détecte si on quitte la fenètre

Nous avons au final une ligne permettant de mettre à jour l'affichage de notre fenêtre: `pygame.display.update()`

Afficher notre labyrinthe

Nous voulons créer un labyrinthe, la première étape est donc de le créer. Pour ceci on va créer un nouveau fichier du nom de **map.txt** et on va définir la map qu'on veut. Exemple:

```
WWWWWWWWWWWWWWWWWWWW
W                  W
W          WWWWWW  W
W    WWWW      W   W
W    W         WWWW W
W  WWW  WWWW       W
W  W W     W       W
W  W       W  WWW WW
W  WWW WWW W W     W
W    W  W    W W   W
WWW  W  WWWWW  W   W
W W    WW          W
W W WWWW    WWW    W
W   W    E    W    W
WWWWWWWWWWWWWWWWWWWW
```

Dans notre cas, les W représentent les murs et le E notre fin. Nous avons besoin de stocker plusieurs informations pour notre labyrinthe, notamment toutes les informations relative aux murs ainsi que l'information relative à la fin de notre niveau. Un simple tableau sera nécessaire à l'initialisation de notre programme:

```
    walls = []
    end_rect = None  # On mets None pour definir qu'il n'y a pas de fin
```

Nos murs seront des classes, une classe est une façon de représenter un objet du monde réel. Par exemple dans notre cas la classe Wall (mur en anglais) permettra de stocker la position, la taille de notre mur et le rectangle le représentant. Au final voici à quoi ressemble notre classe:

```
class Wall:
    def __init__(self, pos):
    walls.append(self)
    self.rect = pygame.Rect(pos[0], pos[1], 32, 32)
```

Il est important de définir notre classe au début de notre programme Ensuite il faut récupérer la map contenue dans le fichier `map.txt` et la stocker, pour cela:

```
map = open("map.txt", "r") # ouvre le fichier 'map.txt' en permettant seul
level = map.readlines() # permet de copier chaque ligne du fichier et les
```

Une fois que nous avons notre level ainsi que le stockage de notre map contenant les murs, nous pouvons tous les créer lors de l'initialisation du programme

```
x = 0
y = 0
for row in level: # Pour chaque ligne dans notre niveau
    for col in row: # Pour chaque colonne (lettre) dans notre ligne
        if col == "W": # Si la lettre est un W
            Wall((x, y)) # Alors on creer un mur aux coordonnees (x, y)
        if col == "E" : # Si la lettre est un E
            end_rect = pygame.Rect(x, y, 32, 32) # Alors on creer le rec
        x += 32 # On ajoute 32 a x (la largeur d  un rectangle)
    y += 32 # On ajoute 32 a y (la hauteur)
    x = 0 # On remet x a 0 pour recommencer au debut de notre ligne
```

Maintenant que nous avons créer tout nos murs nous devons les afficher, est ce que vous vous souvenez de la ligne suivante ? `pygame.display.update()` Et bien modifiez le code pour qu'il ressemble à ça:

```
for  wall  in  walls  :
    pygame.draw.rect(screen, (255, 255, 255), wall.rect)
# pygame.draw.rect(screen, (255, 0, 0), end_rect)
player.draw(screen)
```

Creation de notre personnage

Comme nous avons fait pour nos murs, créons une classe représentant notre Player:

```
class Player:
def __init__(self):
self.rect = pygame.Rect(64, 64, 32, 32) # On creer le rectangle de collisi
self.sprite = pygame.image.load( "res/santa\_top.png" ) # On creer un spri
self.sprite = pygame.transform.scale(self.sprite, (32, 32)) # On change se

def draw(self, screen):
screen.blit(self.sprite, self.rect) # On affiche le sprite
```

Notre classe Player va se charger de toute la création du joueur ainsi que de son affichage. Maintenant que nous avons la définition de notre classe il nous reste plus qu'à créer notre player lors de l'initialisation.

`player = Player()`

Comme on peux le voir, le player utilise une image stockée dans le dossier res et qui s'appelle **santa_top.png**. Ce sont des images qui vous sont données par les encadrants, il faut juste les importer et les stocker dans un dossier appelé res .

Une fois que notre Player est créé, on peux simplement l'afficher grâce à la ligne player.draw(screen). Modifiez le programme pour qu'il ressemble à ça:

```
player.draw(screen)
pygame.display.update()
```

Faire bouger notre personnage

On a besoin que notre personnage bouge,pour ce faire nous allons lui créer une fonction move,il faut savoir que draw est aussi une fonction de notre player, il faut donc rajouter le code suivant dans la classe Player

```
def move(self, dx, dy):
    if dx != 0:
        self.move_single_axis(dx, 0)
    if dy != 0:
        self.move_single_axis(0, dy)
def move_single_axis(self, dx, dy):
# Move the rect
    self.rect.x += dx
    self.rect.y += dy
# If you collide with a wall, move out based on velocity
for wall in walls:
```

```
        if self.rect.colliderect(wall.rect):
            if dx > 0: # Moving right; Hit the left side of the wall
                self.rect.right = wall.rect.left

            '''A VOUS DE COMPLETEZ POUR LES AUTRES DIRECTIONS'''
```

CODE: A VOUS DE COMPLETEZ POUR LES AUTRES DIRECTIONS !

Le code que vous voyez ci-dessus permet de vérifier si dans la direction où nous allons il y a un mur, si jamais il y en a un, on ne bouge pas dans cette direction. Après avoir créé notre fonction move il faut l'appeler, pour ceci on a juste à modifier le code pour qu'il ressemble à ça:

```
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
    # Move the player if an arrowkey is pressed
    '''A COMPLETER'''
```

CODE: Compléter les mouvements du joueur A l'aide de la doc pygame, faites déplacer le joueur en appuyant sur les touches directionnelles

Animer notre personnage

Nous voulons animer notre personnage pour qu'il change de sprite en fonction de la direction dans laquelle il se déplace, pour ce faire nous allons créer une fonction `change_animation` qui va prendre comme paramètre la direction dans laquelle le joueur se déplace.

```
    def change_animation(self, dir):
        self.sprite = pygame.image.load("res/santa_" + dir + ".png")
        self.sprite = pygame.transform.scale(self.sprite, (32, 32))
```

Cette fonction nous allons l'appeler dans la fonction move_single_axis grâce au code suivant:

```
    new_sprite = "left"
    if dx > 0:
    new_sprite = "right"
    elif dx < 0:
    new_sprite = "left"
    elif dy > 0:
```

```
        new_sprite = "bottom"
    elif dy < 0:
        new_sprite = "top"
    self.change_animation(new_sprite)
```

Fin du jeu

Maintenant que nous avons tout ce qu'il faut pour faire bouger notre personnage, il nous faut créer la fin du jeu, pour ce faire nous allons ajouter du code dans la boucle de jeu pour vérifier que le joueur est sur la case de fin.

```
    if player.rect.colliderect(end_rect):
        pygame.quit()
        sys.exit()
```

Fin ?

Et voila, vous avez fini le tutoriel, vous pouvez maintenant vous amuser à créer des niveaux et à les partager avec vos amis. Voici une liste de bonus que vous pouvez faire si vous avez le temps:

- Créer un menu
- Créer un écran de victoire
- Créer un écran de défaite
- Créer un écran de pause
- Créer un écran de sélection de niveau
- Créer un écran de création de niveau
- Créer un écran de paramètres
- Et plein d'autres choses encore