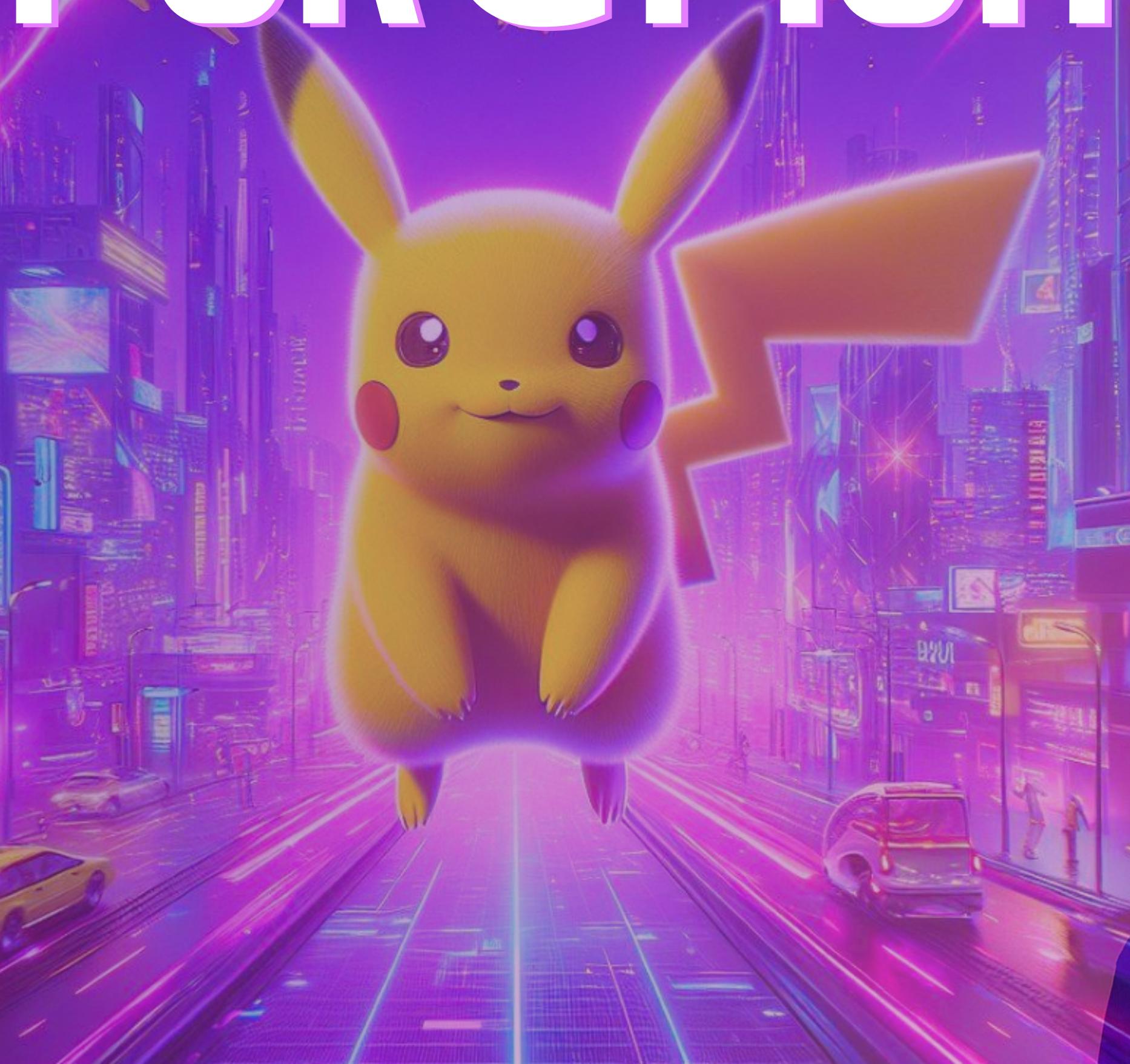
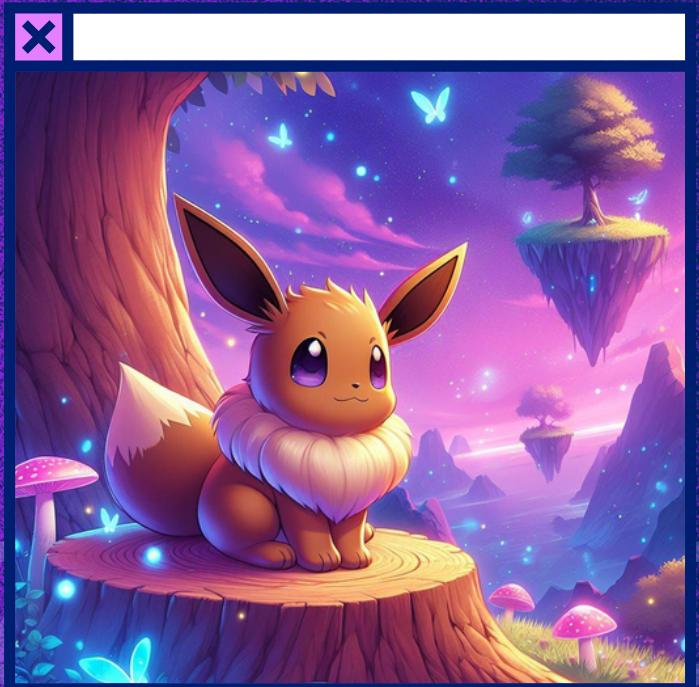




# POKÉMON



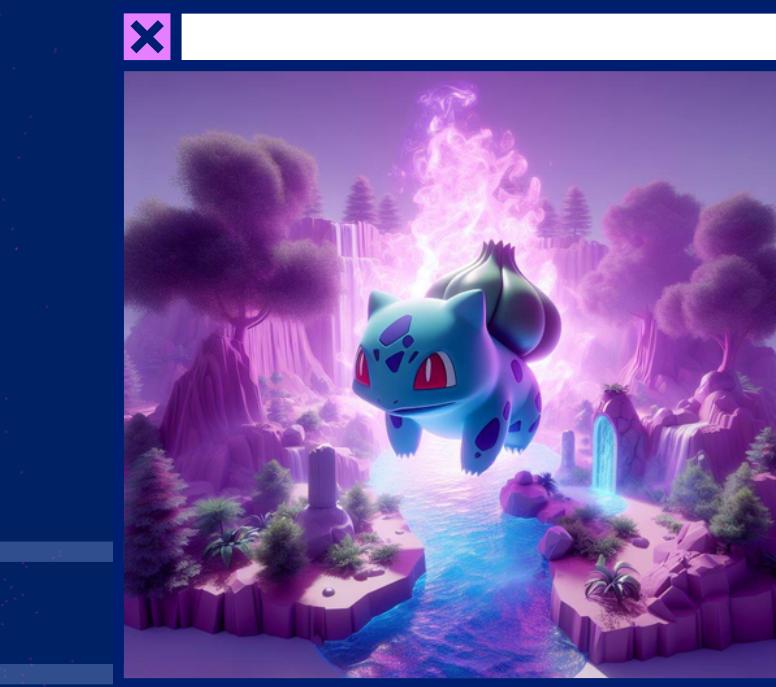
# MEET THE TEAM



Maysa



Kamelia



Sofia



# AGENDA

- 1** Présentation du Projet sur Trello et GitHub
- 2** Partie Maysa : Page Menu et Map
- 3** Partie Kamelia : Page Combat et Page Pokémon
- 4** Partie Sophia : Page Pokédex



Two side-by-side screenshots of a Trello board titled "Pokémon".

**Left Screenshot:**

- Labels:** les classes, En cours, Terminé, presque fini, note importante, noms git.
- Items:**
  - les classes: Planning du projet, Réunion de lancement, class Combat : méthode récupérée => récupérée la puissance de l'adversaire, méthode enlève : points de vie en fonction de la défense, méthode renvoie : nom de vainqueur / perdant ou gagnant, méthode enregistrer : Pokémon rencontré dans votre Pokédex. ( Kamelia )
  - En cours: le menu en cours, création des branches, les branch du github, class Pokedex : outil utiliser pour enregistrer les informations sur les Pokémon rencontrés.
  - Terminé: class Pokemon : Présentation des 5 Pokémons qu'on a choisis. Effectuer des actions.
  - presque fini: la page menu, + Ajouter une carte
  - note importante: faut Jamais oublier de faire des commit, + Ajouter une carte
  - noms git: git add .git commit .git push origin nom de la branche que tu veux .git switch (l nom de la branche) .git pull pour tout récupérer si tu veux merger tu fait git merge le nom de la branche, + Ajouter une carte
- Buttons:** + Ajouter une carte

**Right Screenshot:**

- Labels:** les classes, En cours, Terminé, presque fini, note importante, noms git.
- Items:**
  - les classes: Planning du projet, Réunion de lancement, class Combat : méthode récupérée => récupérée la puissance de l'adversaire, méthode enlève : points de vie en fonction de la défense, méthode renvoie : nom de vainqueur / perdant ou gagnant, méthode enregistrer : Pokémon rencontré dans votre Pokédex. ( Kamelia )
  - En cours: le menu en cours, création des branches, les branch du github, class Pokedex : outil utiliser pour enregistrer les informations sur les Pokémon rencontrés.
  - Terminé: class Pokemon : Présentation des 5 Pokémons qu'on a choisis. Effectuer des actions.
  - presque fini: la page menu, + Ajouter une carte
  - note importante: faut Jamais oublier de faire des commit, + Ajouter une carte
  - noms git: git add .git commit .git push origin nom de la branche que tu veux .git switch (l nom de la branche) .git pull pour tout récupérer si tu veux merger tu fait git merge le nom de la branche, + Ajouter une carte
- Buttons:** + Ajouter une autre liste

A screenshot of a GitHub repository's branches page.

**Branches Overview:**

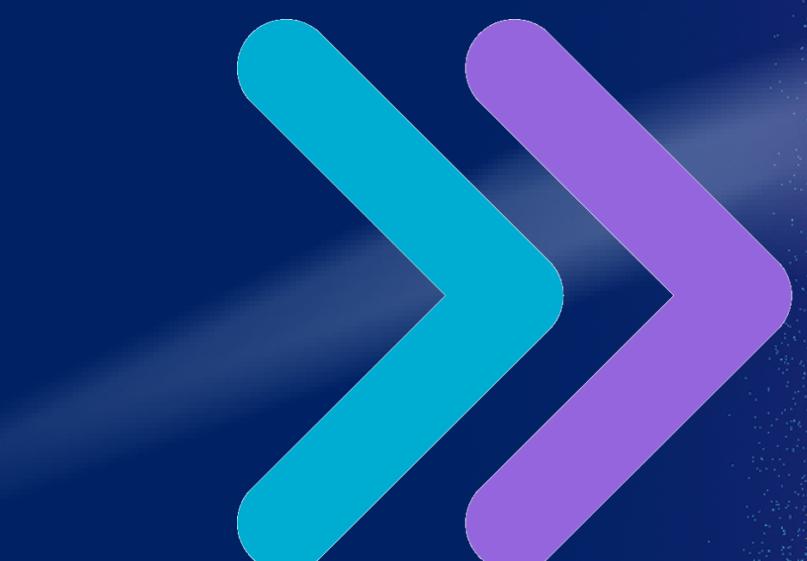
Branch	Updated	Check status	Behind	Ahead	Pull request
main	last month	Default	0	113	...

**Your branches:**

Branch	Updated	Check status	Behind	Ahead	Pull request
Maysa	3 hours ago	0/113	0	113	...
kamelia	2 days ago	0/103	0	103	...
sofia	last week	0/113	0	113	...

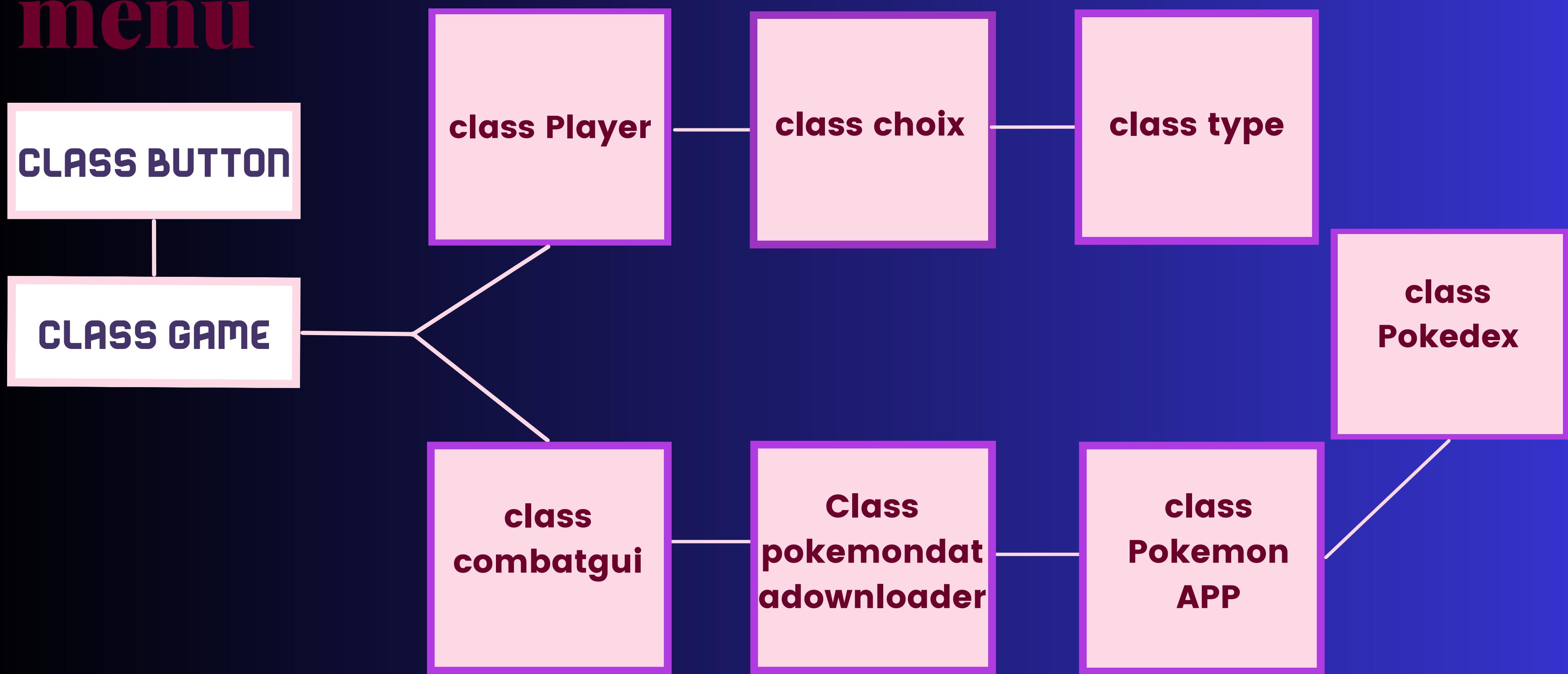
**Active branches:**

Branch	Updated	Check status	Behind	Ahead	Pull request
Maysa	3 hours ago	0/113	0	113	...
kamelia	2 days ago	0/103	0	103	...
sofia	last week	0/113	0	113	...



# Diagram des classes

## menu



# Page Menu



- Importer les modules nécessaires pour mon page
- Définition des fonctions pour mon code .

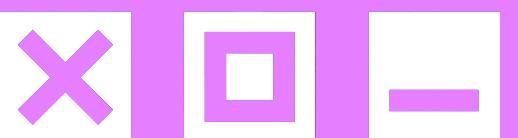
```
fichier > Menu.py > main_menu
1 import pygame
2 import sys
3 from button import Button
4 from game import *
5 from game1 import Game
6 from pokemon import PokemonApp
7 from combat import load_pokedex
8
9
10 pygame.init()
11 SCREEN = pygame.display.set_mode((900, 750))
12 pygame.display.set_caption("Menu")
13
14 BG = pygame.image.load("photos/_005551e8-0912-4606-bc6e-c497483f9886.jpg")
15
16 pokemons = load_pokedex()
17
18 > def get_font(size): ...
19 > def Lancer_une_partie(): ...
20
21
22 > if __name__ == "__main__":
23     main_menu()
```



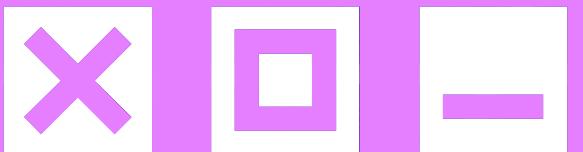
```
> def Lancer_une_partie(): ...
>
> def Pokemon(): ...
>
> def Pokédex(): ...
>
> def main_menu(): ...
>
> if __name__ == "__main__":
>     main_menu()
```

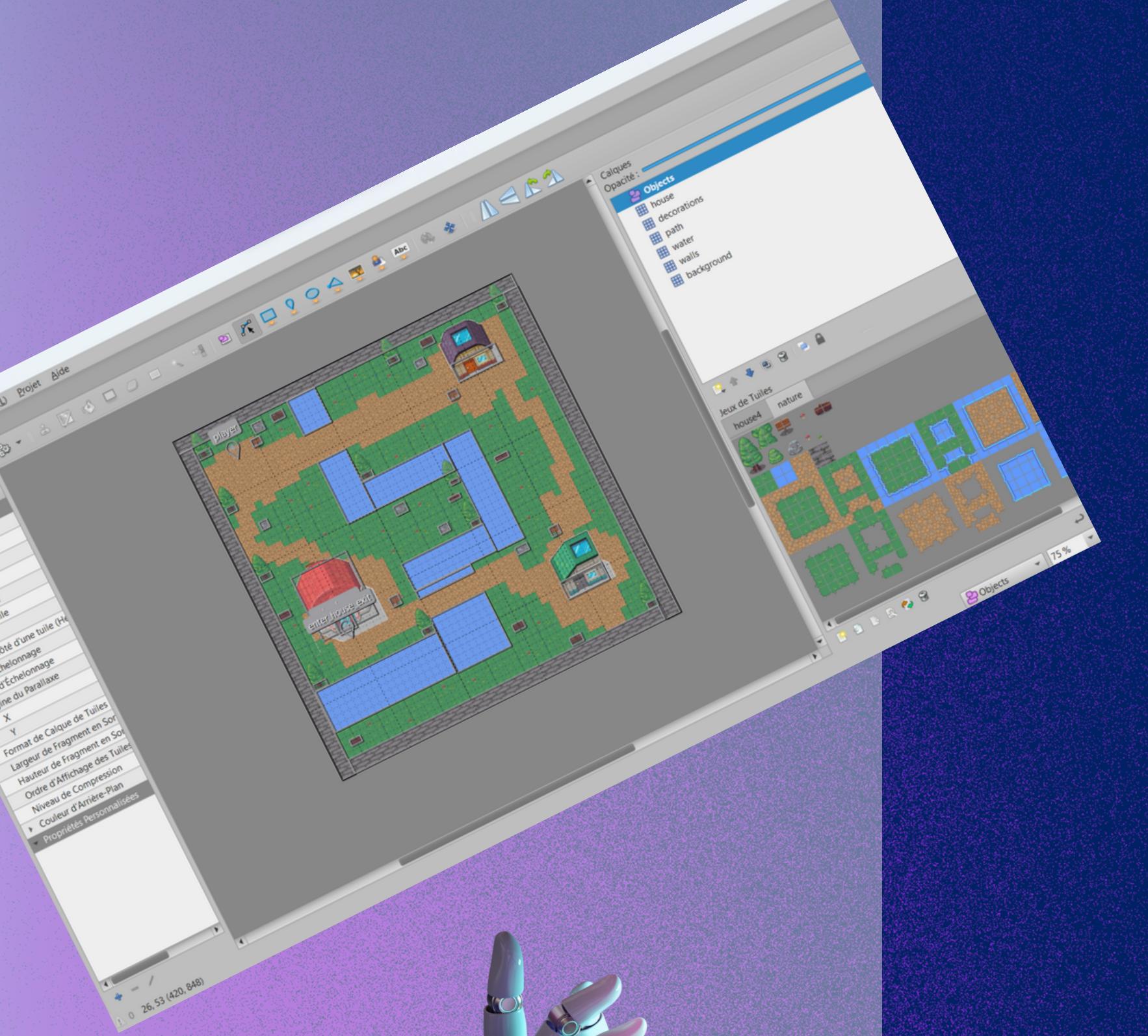
# BUTTONS

Puis créer cette class  
qui me permet de  
modifier (position,  
colors, .... ) 



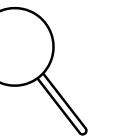
```
fichier > button.py > Button
1  class Button:
2      def __init__(self, pos, text_input, font, base_color, hovering_color):
3          # Initialisation de la classe Button
4          self.pos = pos # Position du bouton
5          self.text_input = text_input # Texte affiché sur le bouton
6          self.font = font # Police d'écriture utilisée pour le texte
7          self.base_color = base_color # Couleur de base du bouton
8          self.hovering_color = hovering_color # Couleur lorsque la souris survole le bouton
9          self.text = self.font.render(self.text_input, True, self.base_color) # Surface contenant le texte rendu avec
10         self.rect = self.text.get_rect(center=self.pos) # Rectangle englobant le texte pour la détection des collisi
11
12     def update(self, window):
13         # Met à jour l'affichage du bouton sur la fenêtre donnée
14         window.blit(self.text, self.rect)
15
16     def checkForInput(self, position):
17         # Vérifie si la position donnée est à l'intérieur du rectangle du bouton
18         if self.rect.collidepoint(position):
19             return True # La position est à l'intérieur du bouton
20         return False # La position n'est pas à l'intérieur du bouton
21
22     def changeColor(self, position):
23         # Change la couleur du texte du bouton si la position donnée est à l'intérieur du rectangle du bouton
24         if self.rect.collidepoint(position):
25             self.text = self.font.render(self.text_input, True, self.hovering_color) # Change la couleur du texte à
26         else:
27             self.text = self.font.render(self.text_input, True, self.base_color) # Reviens à la couleur de base du t
```





# MAP

• Tiled



- Terminer le menu.
- Une idée pour créer une carte. 🧠💡
- Réfléchir Bien, voir des vidéos.

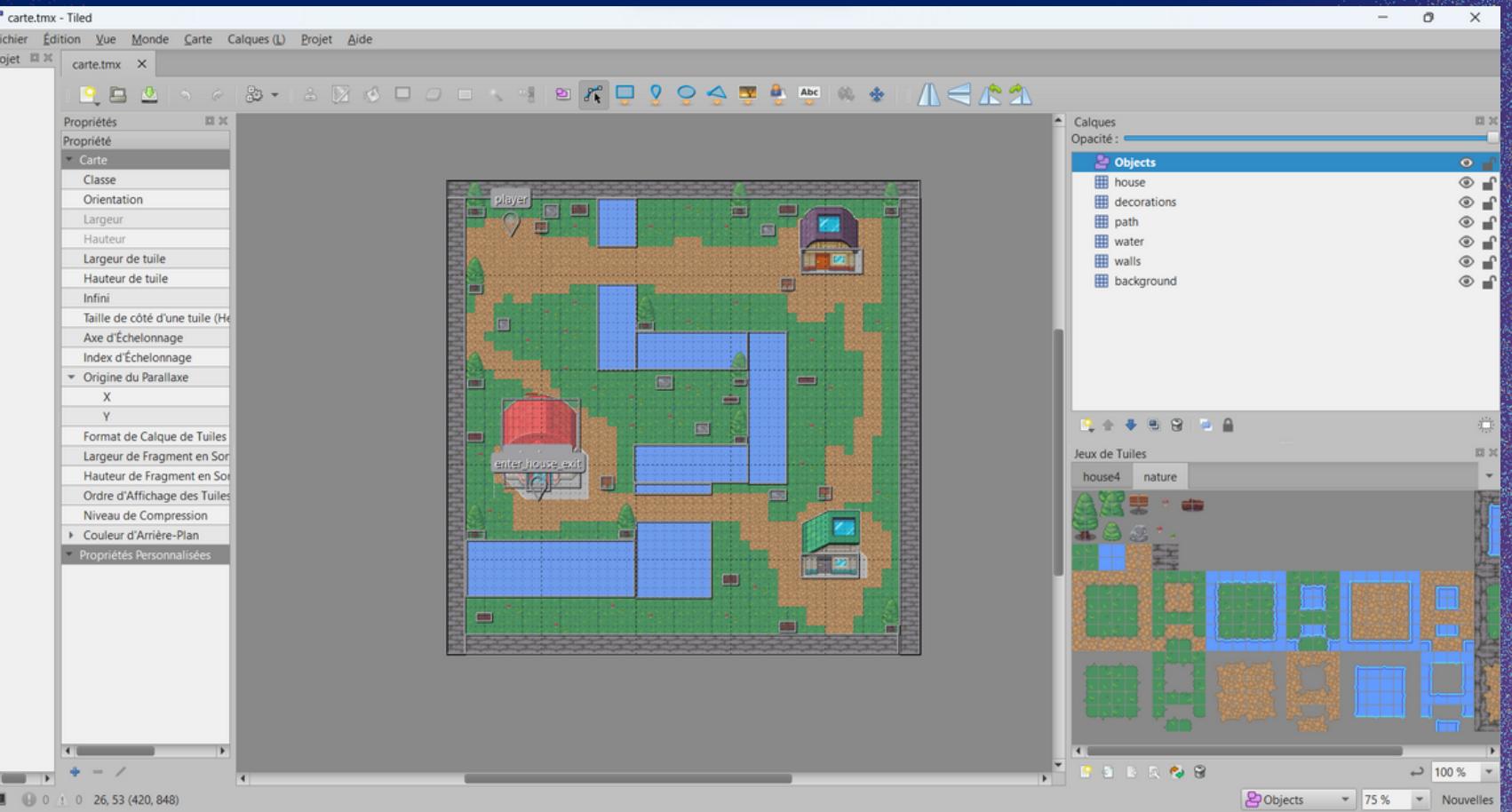
NEXT

# MAP

player, game1

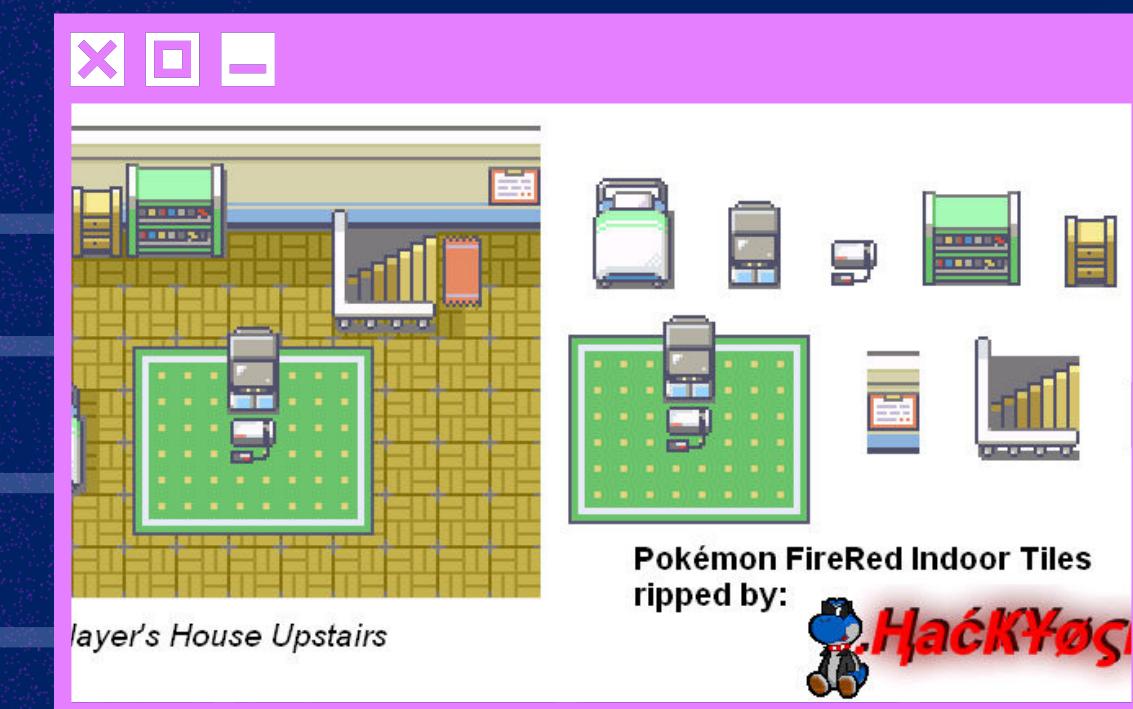


- on fait étape par étape le carte, ( dessiner : background, murs, d'eau, chemin, décoration, maisons, objects).
- on est décidé faire un player.
- on mit le page game1 qui fonction la carte .



comment ??





# • comment j'ai crée la carte étape par étape à l'aide de Tiled

```
✓ Map
  > __pycache__
  carte.tmx
  house 1.png
  house 1.tsx
  house.tmx
  house1.tmx
  house2.png
  house2.tsx
  house3.jpg
  house3.tsx
  house4.png
  house4.tsx
  nature.tsx
  natureset.png
  player.png
  > photos
  {} pokedex.json
  {} pokemon.json
  ⓘ README.md
  > OUTLINE
```

```
5  from pokemont import PokemonApp
6  from player import Player # Importation de la classe Player
7  from combat import CombatGUI, get_random_rival, load_pokemon
8  from pygame.locals import K_UP, K_DOWN, K_LEFT, K_RIGHT
9  # Classe principale représentant le jeu Pokémons
10 class Game:
11     def __init__(self):
12         # créer la fenêtre du jeu
13         self.screen = pygame.display.set_mode((900, 750))
14         pygame.display.set_caption('Pokemon')
15         # Attribut pour suivre la carte actuelle ('world' ou 'city')
16         self.map = 'world'
17         # Chargement de la carte TMX
18         tmx_data = pytmx.util_pygame.load_pygame('Map/carte.tmx')
19         map_data = pyscroll.data.TiledMapData(tmx_data)
20         map_layer = pyscroll.orthographic.BufferedRenderer(map_data)
21         map_layer.zoom = 1
22
23         # générer un joueur
24         player_position = tmx_data.get_object_by_name('player')
25         self.player = Player(player_position.x, player_position.y)
26         # Initialisation de la liste des rectangles de collision
27         # définir une liste qui va stocker les rectangles
28         self.walls = []
29         # Recherche des objets de collision dans la carte
```



# CLASS GAME : PYGAME

- 1 - DEF INIT
- 2 - DEF HANDLE\_INPUT
- 3 - DEF SWITCH\_HOUSE
- 4 - DEF SWITCH\_WORLD
- 5 - DEF UPDATE
- 6 - DEF RUN



```
1 import pygame
2 import sys
3 import pytmx
4 import pyscroll
5 from player import Player
6 from combat import CombatGUI, get_random_rival, load_pokedex, select_pokemon_screen
7 from pokemon import PokemonApp
8 from pygame.locals import K_UP, K_DOWN, K_LEFT, K_RIGHT
9 # Classe principale représentant le jeu Pokémon
10 class Game:
11     >     def __init__(self): ...
12
13
14     # tout le clay pour le joueur
15     >     def handle_input(self): ...
16
17
18     # Méthode pour changer de carte vers l'intérieur de la maison
19     >     def switch_house(self): ...
20
21
22     >     def switch_world(self): ...
23
24
25     >     def start_combat(self): ...
26
27
28     >     def update(self): ...
29
30
31     >     def run(self): ...
```

# CLASS PLAYER

comment !!



Utilisation de la bibliothèque pygame :

- Initialisation : Chargement des sprites, configuration des animations et des rectangles de collision, initialisation des variables de position et de vitesse.
- Méthode **update** : Mise à jour de la position du joueur et de son rectangle de collision.
- Méthode **move\_back** : Gestion des collisions et retour à la position précédente.

```
fichier > player.py > Player > __init__  
3  
4     # Classe représentant le joueur contrôlé par l'utilisateur  
5     class Player(pygame.sprite.Sprite):  
6         >     def __init__(self, x, y): ...  
26  
27         # Méthode pour enregistrer la position actuelle du joueur  
28         >     def save_location(self): ...  
30  
31         # Méthode pour changer l'animation du joueur  
32         >     def change_animation(self, name): ...  
35  
36         # Méthodes pour les mouvements du joueur dans différentes directions  
37         >     def move_right(self): ...  
39  
40         >     def move_left(self): ...  
42  
43         >     def move_up(self): ...  
45  
46         >     def move_down(self): ...  
48  
49         # Méthode pour mettre à jour la position du joueur et son rectangle de collision  
50         >     def update(self): ...  
53  
54         # Méthode pour revenir à la position précédente en cas de collision  
55         >     def move_back(self): ...  
59  
60         # Méthode pour extraire une image de la feuille de sprites du joueur  
61         >     def get_image(self, x, y): ...  
65
```



Résultat



## 2 Idée



Créer un fonction qui permet au player de rentrer à la maison .



```
X  S  M

def update(self):
    self.group.update()

    # vérifier l'entrer dans la maison
    if self.player.feet.colliderect(self.enter_house_rect):
        self.switch_house()

    # vérifier l'entrer dans la maison
    if self.map == 'world' and self.player.feet.colliderect(self.enter_house_rect):
        self.switch_house()
        self.map = 'house'

    # vérifier l'entrer dans la maison
    if self.map == 'house' and self.player.feet.colliderect(self.enter_house_rect):
        self.switch_world()
        self.map = 'world'

    # verification collision
    for sprite in self.group.sprites():
        if sprite.feet.collidelist(self.walls) > -1:
            sprite.move_back() # revenir à position avant le déplacement
```

### def switch\_house



# vérifier l'entrer dans la maison  
if self.player.feet.colliderect(self.enter\_house\_rect):  
 self.switch\_house()



# DEF SWITCH\_WORLD



c'est quoi ?



La méthode qui permet  
au player de sortir de la  
maison.

Mais il se passe quoi ? 🤔



```
self.player.position[1] = spawn_mouse_point.y + 50

# Afficher un bouton pour démarrer le combat
button_font = pygame.font.Font("photos/Pokemon Solid.ttf", 45)
button_text = button_font.render("Start Combat", True, (255, 215, 0))
button_rect = button_text.get_rect(center=(450, 375))
button_font = pygame.font.Font("photos/Pokemon Solid.ttf", 45)
return_button_text = button_font.render("Return to Map", True, (255, 215, 0))
return_button_rect = return_button_text.get_rect(center=(450, 475))

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
        elif event.type == pygame.MOUSEBUTTONDOWN:
            mouse_pos = pygame.mouse.get_pos()
            if button_rect.collidepoint(mouse_pos):
                self.start_combat()
            elif return_button_rect.collidepoint(mouse_pos):
                return # Retour à la carte

    self.screen.fill((129, 178, 154))
    self.screen.blit(button_text, button_rect)
    self.screen.blit(return_button_text, return_button_rect)
    pygame.display.flip()
```

# FINALEMENT

Une fonction qui me permet de choisir entre commencer le combat ou return à la carte



Start Combat  
Return to Map



```
def switch_house(self):
    (variable) tmx_data: Any
        # la maison
    tmx_data = pytmx.util_pygame.load_pygame("Map/house.tmx")
    map_data = pyscroll.data.TiledMapData(tmx_data)
    map_layer = pyscroll.orthographic.BufferedRenderer(map_data, self.screen.get_size())
    map_layer.zoom = 1

    # Réinitialisation de la liste des rectangles de collision
    self.walls = []

    for obj in tmx_data.objects:
        if obj.type == "collision":
            self.walls.append(pygame.Rect(obj.x, obj.y, obj.width, obj.height))

    # dessiner le groupe de calques avec le joueur dans la maison
    self.group = pyscroll.PyscrollGroup(map_layer=map_layer, default_layer=6)
    self.group.add(self.player)

    # definir le rect de collision pour sortir de la maison
    enter_house = tmx_data.get_object_by_name('exit_house')
    self.enter_house_rect = pygame.Rect(enter_house.x, enter_house.y, enter_house.width, enter_house.height)

    # recuperer du joueur dans la maison
    spawn_house_point = tmx_data.get_object_by_name('spawn_house')
    self.player.position[0] = spawn_house_point.x
    self.player.position[1] = spawn_house_point.y - 30
```

```
def switch_world(self):
    # charger la carte (tmx)
    tmx_data = pytmx.util_pygame.load_pygame("Map/carte.tmx")
    map_data = pyscroll.data.TiledMapData(tmx_data)
    map_layer = pyscroll.orthographic.BufferedRenderer(map_data, self.screen.get_size())
    map_layer.zoom = 1

    # definir une liste qui va stocker les rectangle
    self.walls = []

    for obj in tmx_data.objects:
        if obj.type == "collision":
            self.walls.append(pygame.Rect(obj.x, obj.y, obj.width, obj.height))

    # dessiner le groupe de calques avec le joueur dans le monde
    self.group = pyscroll.PyscrollGroup(map_layer=map_layer, default_layer=6)
    self.group.add(self.player)

    # definir le rect de collision pour entrer devant la maison
    enter_house = tmx_data.get_object_by_name('enter_house')
    self.enter_house_rect = pygame.Rect(enter_house.x, enter_house.y, enter_house.width, enter_house.height)

    # recuperer
    spawn_house_point = tmx_data.get_object_by_name('enter_house_exit')
    self.player.position[0] = spawn_house_point.x
    self.player.position[1] = spawn_house_point.y + 30
```

```
def update(self):
    self.group.update()

    # verifier l'entrer dans la maison
    if self.player.feet.colliderect(self.enter_house_rect):
        self.switch_house()

    # verifier l'entrer dans la maison
    if self.map == 'world' and self.player.feet.colliderect(self.enter_house_rect):
        self.switch_house()
        self.map = 'house'

    # verifier l'entrer dans la maison
    if self.map == 'house' and self.player.feet.colliderect(self.enter_house_rect):
        self.switch_world()
        self.map = 'world'

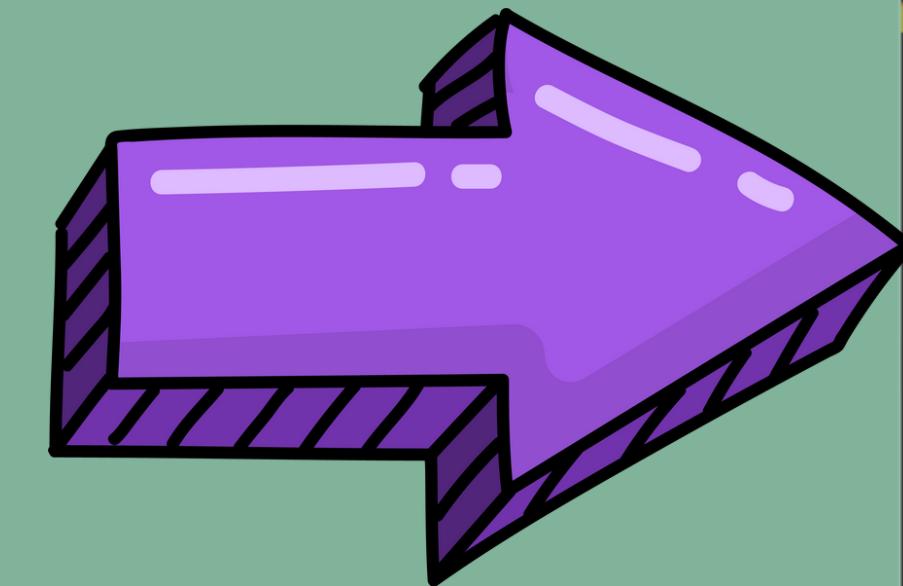
    # verification collision
    for sprite in self.group.sprites():
        if sprite.feet.collidelist(self.walls) > -1:
            sprite.move_back() # revenir a position avant le deplacement
```

# CHOIX

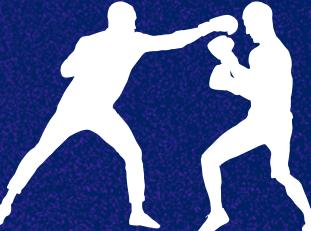
NEXT

```
1 import pygame
2 import requests # Ajout de l'import pour requests
3 import io
4 from urllib.request import urlopen
5
6 base_url = 'https://pokeapi.co/api/v2' # Définition de base_url
7
8 class Type:
9     def __init__(self, type_name):
10         self.type_name = type_name
11
12 class choix(pygame.sprite.Sprite):
13     def __init__(self,nom, points_de_vie, niveau, puissance_attaque, defense, types, x, y):
14         pygame.sprite.Sprite.__init__(self)
15         self.nom = nom
16         self.points_de_vie = points_de_vie
17         self.max_points_de_vie = points_de_vie
18         self.puissance_attaque = puissance_attaque
19         self.defense = defense
20         self.type = [Type(type_name) for type_name in types]
21         self.niveau = niveau
22         self.x = x
23         self.y = y
24
25     def set_sprite(self, side):
26         req = requests.get(f"{base_url}/pokemon/{self.nom.lower()}")
27         self.json = req.json()
28
29         image = self.json['sprites'][side]
30         image_stream = urlopen(image).read()
31         image_file = io.BytesIO(image_stream)
32         self.image = pygame.image.load(image_file).convert_alpha()
33
34         scale = self.size / self.image.get_width()
35         new_width = self.image.get_width() * scale
36         new_height = self.image.get_height() * scale
37         self.image = pygame.transform.scale(self.image, (int(new_width), int(new_height)))
38
39     def draw(self, game, x, y, alpha=255):
40         sprite = self.image.copy()
41         transparency = (255, 255, 255, alpha)
42         sprite.fill(transparency, None, pygame.BLEND_RGBA_MULT)
43         game.blit(sprite, (x, y))
44
45     def get_rect(self):
46         return pygame.Rect(self.x, self.y, self.image.get_width(), self.image.get_height())
47
48
49
50
51
```

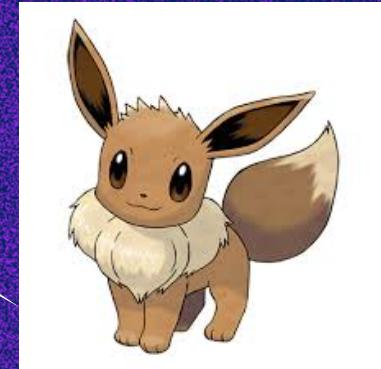
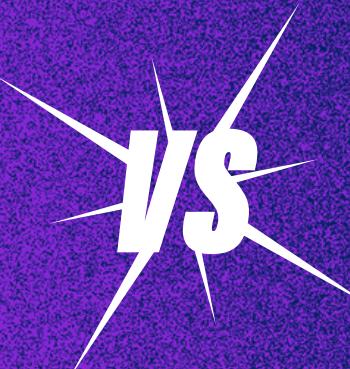
1. Importé la bibliothèque `pygame` ainsi que les modules `requests`, `io`, et `urlopen`.
2. Défini la variable `base_url` contenant l'URL de base de l'API `PokeAPI`.
3. Crée la classe `Type` qui représente les différents types de Pokémons.
4. Crée la classe `choix`, une sous-classe de `pygame.sprite.Sprite`, qui représente un Pokémon sélectionné par le joueur.
5. Défini la méthode `__init__` pour initialiser les attributs du Pokémon.
6. Implémenté la méthode `set_sprite` pour charger l'image du Pokémon à partir de l'API `PokeAPI` en utilisant `requests` et `urlopen`.
7. Défini la méthode `draw` pour dessiner le Pokémon sur l'écran de jeu.
8. Implémenté la méthode `get_rect` pour obtenir un objet `pygame.Rect` représentant la zone occupée par le sprite du Pokémon sur l'écran.



# COMBAT

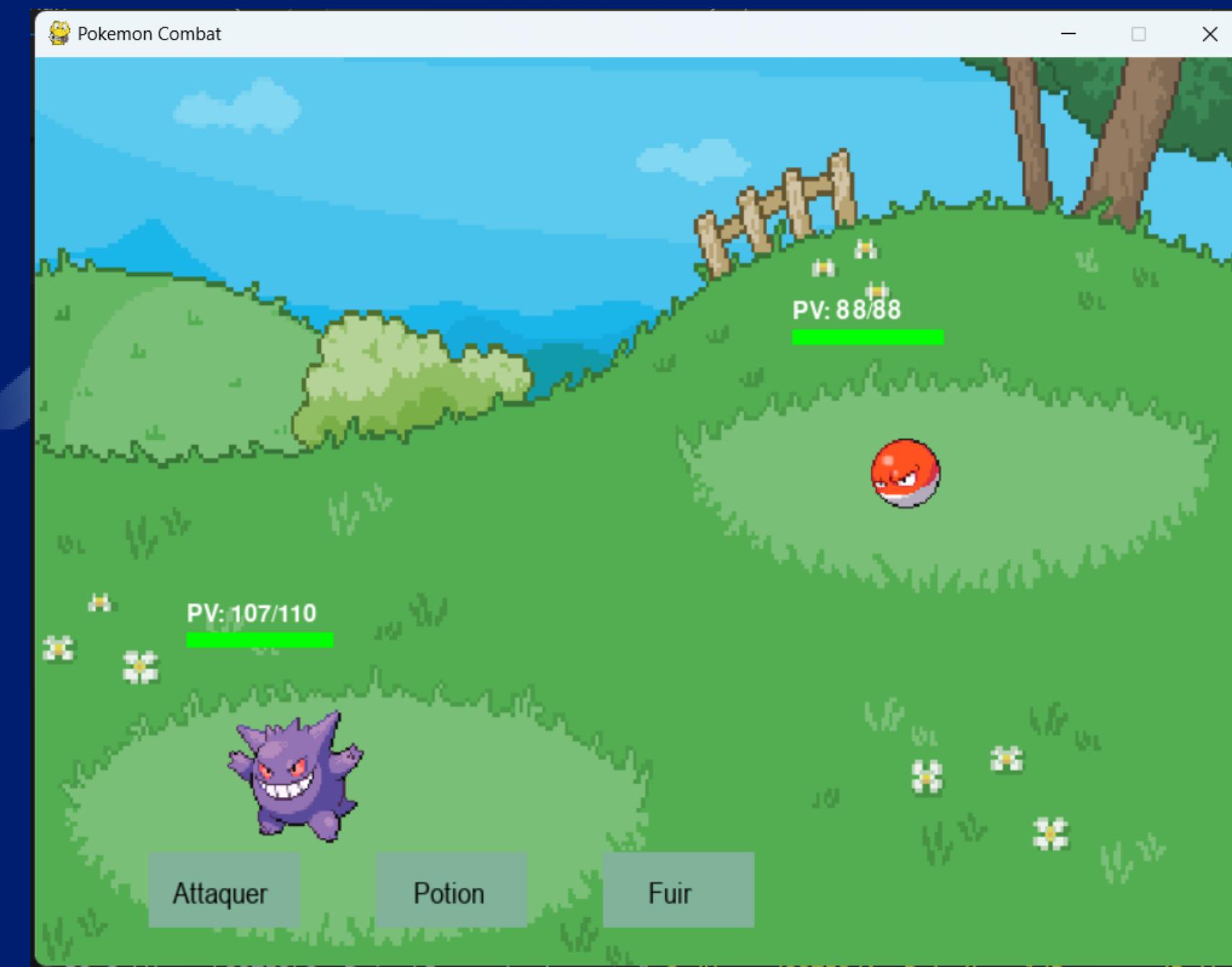


1. Chargement des données des Pokémons à partir d'un fichier JSON.
2. Sélection aléatoire d'un Pokémon rival parmi une liste de Pokémons chargée.
3. Affichage de l'écran de sélection du Pokémon du joueur.
4. Gestion des événements de clic de souris pour sélectionner le Pokémon du joueur.
5. Calcul des dégâts infligés par les Pokémons lors du combat en fonction de leurs caractéristiques.
6. Application des dégâts aux Pokémons adverses.
7. Affichage de l'état de santé des Pokémons et des messages de combat.
8. Utilisation de potions pour restaurer les points de vie du Pokémon du joueur.
9. Détermination du vainqueur du combat.
10. Enregistrement du Pokémon perdant dans le Pokédex.



```
Menu.py    choix.py    game.py    combat.py X    pokemon.py    pok.py    pokemon.json    g    D    S ...  
fichier > combat.py > select_pokemon_screen  
1  import pygame # Importation de la bibliothèque Pygame pour créer l'interface graphique  
2  from urllib.request import urlopen # Importation de la fonction urlopen pour effectuer des requêtes HTTP  
3  import json # Importation du module json pour travailler avec des données JSON  
4  import random # Importation du module random pour générer des nombres aléatoires  
5  import time # Importation du module time pour gérer le temps dans le jeu  
6  from choix import * # Importation de la classe 'choix', mais elle n'est pas utilisée dans ce code  
7  
8  # Définition des constantes de couleur pour une utilisation ultérieure dans le code  
9  black = (0, 0, 0)  
10 white = (255, 255, 255)  
11  
12 # URL de base pour l'API des Pokémons, mais non utilisée dans ce code  
13 base_url = 'https://pokeapi.co/api/v2'  
14  
15 # Initialisation de Pygame et chargement de la musique de fond  
16 pygame.mixer.init()  
17 pygame.mixer.music.load('photos/PkmRB-Enc2.mid')  
18 pygame.mixer.music.play(-1)  
19  
20 # Définition de la classe Type (représentant les types de Pokémons)  
21 > class Type: ...  
22  
23 # Définition de la classe CombatGUI pour gérer l'interface du combat Pokémons  
fichier > combat.py > CombatGUI  
24  
25 > class Type: ...  
26  
27 # Définition de la classe CombatGUI pour gérer l'interface du combat Pokémons  
28 > class CombatGUI: ...  
29  
30 # Fonction pour charger les données des Pokémons à partir d'un fichier JSON  
31 > def load_pokedex(): ...  
32  
33 # Fonction pour obtenir un Pokémon rival aléatoire parmi une liste de Pokémons  
34 > def get_random_rival(pokemons, player_pokemon): ...  
35  
36 # Fonction pour afficher l'écran de sélection du Pokémon du joueur  
37 > def select_pokemon_screen(game, pokemons, K): ...  
38  
39  
40
```

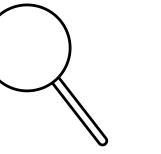
# combat



# Pok



# Pokémon



Initialisation.

Téléchargement des images.

Récupération des données des Pokémons.

Enregistrement des données.

Exemple d'utilisation

Initialisation de l'application et de la fenêtre Pygame .

Gestion des événements.

Recherche de Pokémons.

Affichage de la liste des Pokémons.

Affichage des détails d'un Pokémons sélectionné.

Boucle principale de l'application.

```
fichier > pok.py > ...
1 import requests
2 import json
3 import os
4
5 > class PokemonDataDownloader:...
47
48 # Exemple d'utilisation de la classe
49 if __name__ == "__main__":
50     downloader = PokemonDataDownloader()
51     downloader.get_pokemon_data()
52
53
54
```

```
fichier > pokemon.py > ...
1 import pygame
2 from pygame.locals import QUIT, KEYDOWN
3 import json
4 import requests
5 from pok import PokemonDataDownloader
6
7 > class PokemonApp:...
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
```



Voltorb

Bulbasaur

Charmander

Squirtle

Sandshrew

Pikachu

Eevee

Wartortle

Staryu

Ivysaur

Gengar

Ajouter

The interface includes a list of Pokémons on the left and three 3D models of Charmander, Pikachu, and Bulbasaur on the right. A purple arrow points from the 'Ajouter' button at the top to the '50' value in the input field below it. Another purple arrow points from the 'Ajouter' button at the bottom right to the 'Diglett' entry in the list.

- Voltorb
- Bulbasaur
- Charmander
- Squirtle
- Sandshrew
- Pikachu
- Eevee
- Wartortle
- Staryu
- Ivysaur
- Gengar
- Diglett

50

Ajouter

Voltorb

Bulbasaur

Charmander

Squirtle

Sandshrew

Pikachu

Eevee

Wartortle

Staryu

Ivysaur

Gengar

Ajouter

Name: bulbasaur  
Types: ['grass', 'poison']  
Hp: 45  
Attack: 49  
Defense: 49  
Speed: 45

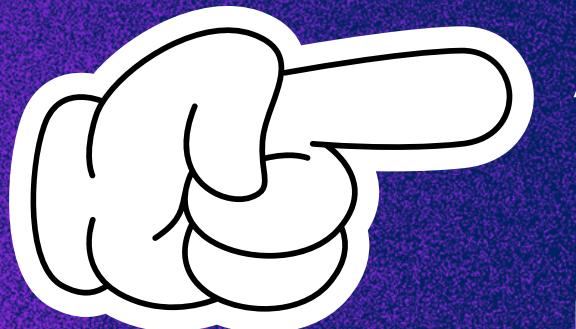


# POKEDEX

.json

*Les renseignements sont complet et y est inclus les photos des pokémons.*

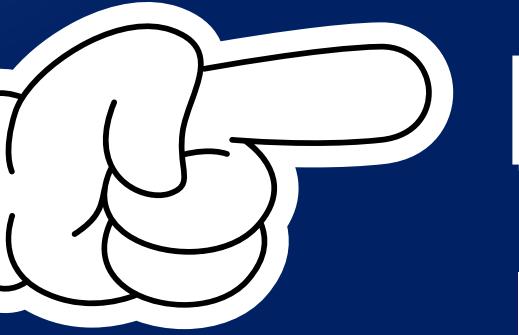
*Il y a 23 pokémons.*



pokedex

Code utilisable pour le combat.

A chaque rencontre sur le combat , le pokémon rencontré est ajouté au fichier.



pokedexgui

Interface graphique pour afficher les pokémons sur le pokedex

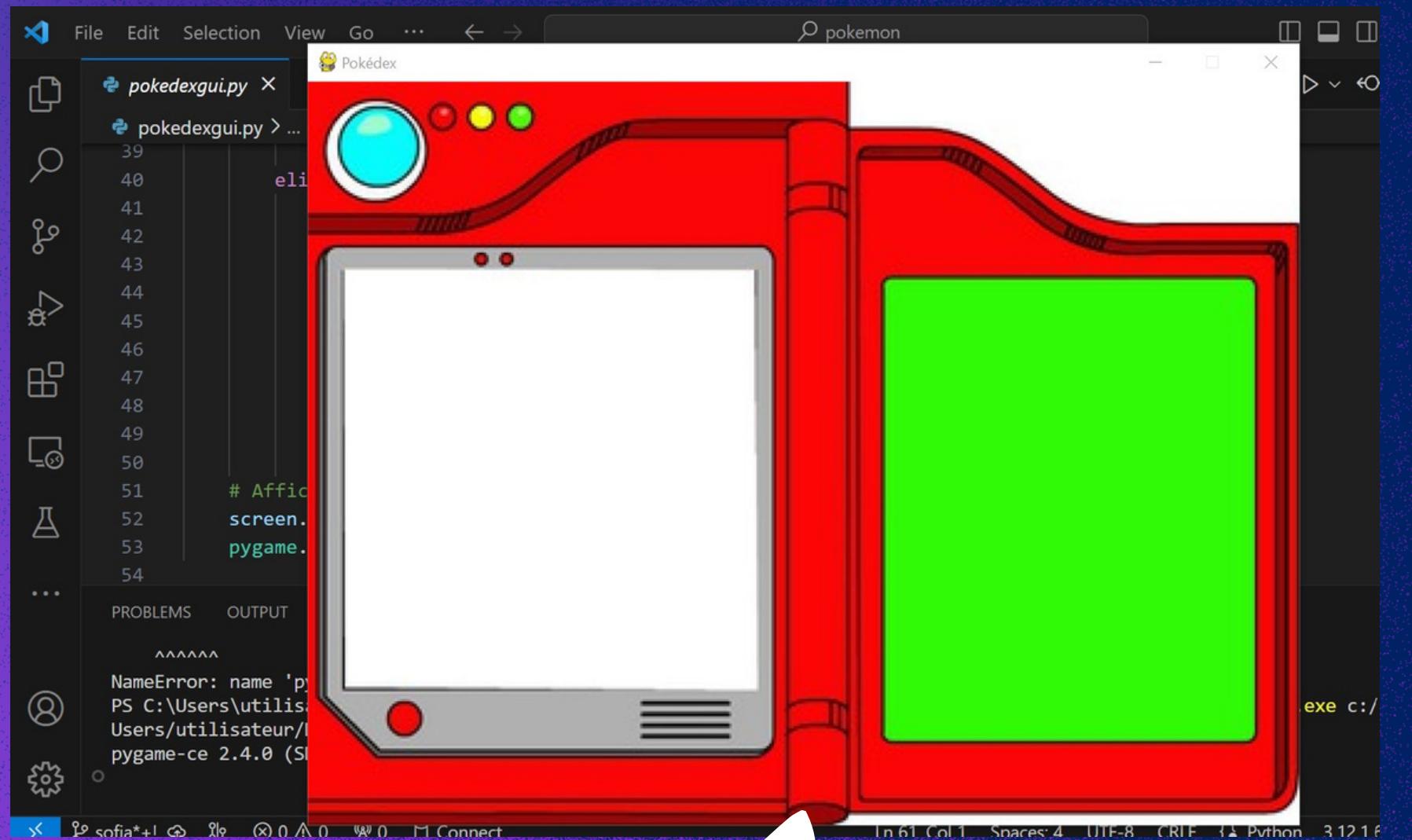
# POKEDEX CE QUI NE FONCTIONNE PAS

Je n'ai pas réussi à fusionner les 2 fichiers .py et à afficher les pokemons sur l'interface graphique.

Mon code ne peut pas être utilisé par les autres pour faire un combat.



# Interface graphique



```
File Edit Selection View Go ... < > pokémon
```

```
pokedexgui.py X
```

```
pokedexgui.py > ...
```

```
39
40
41
42
43
44
45
46
47
48
49
50
51     # Affiche le pokédex
52     screen =
53     pygame.
54
```

PROBLEMS OUTPUT

^^^^^  
NameError: name 'pygame' is not defined  
PS C:\Users\utilisateur\PycharmProjects\pokedex> python pokedexgui.py

```
In 61 Col 1 Spaces: 4 UTF-8 CR LF 32 Python 3.12.1
```

```
File Edit Selection View Go ... < > pokémon
```

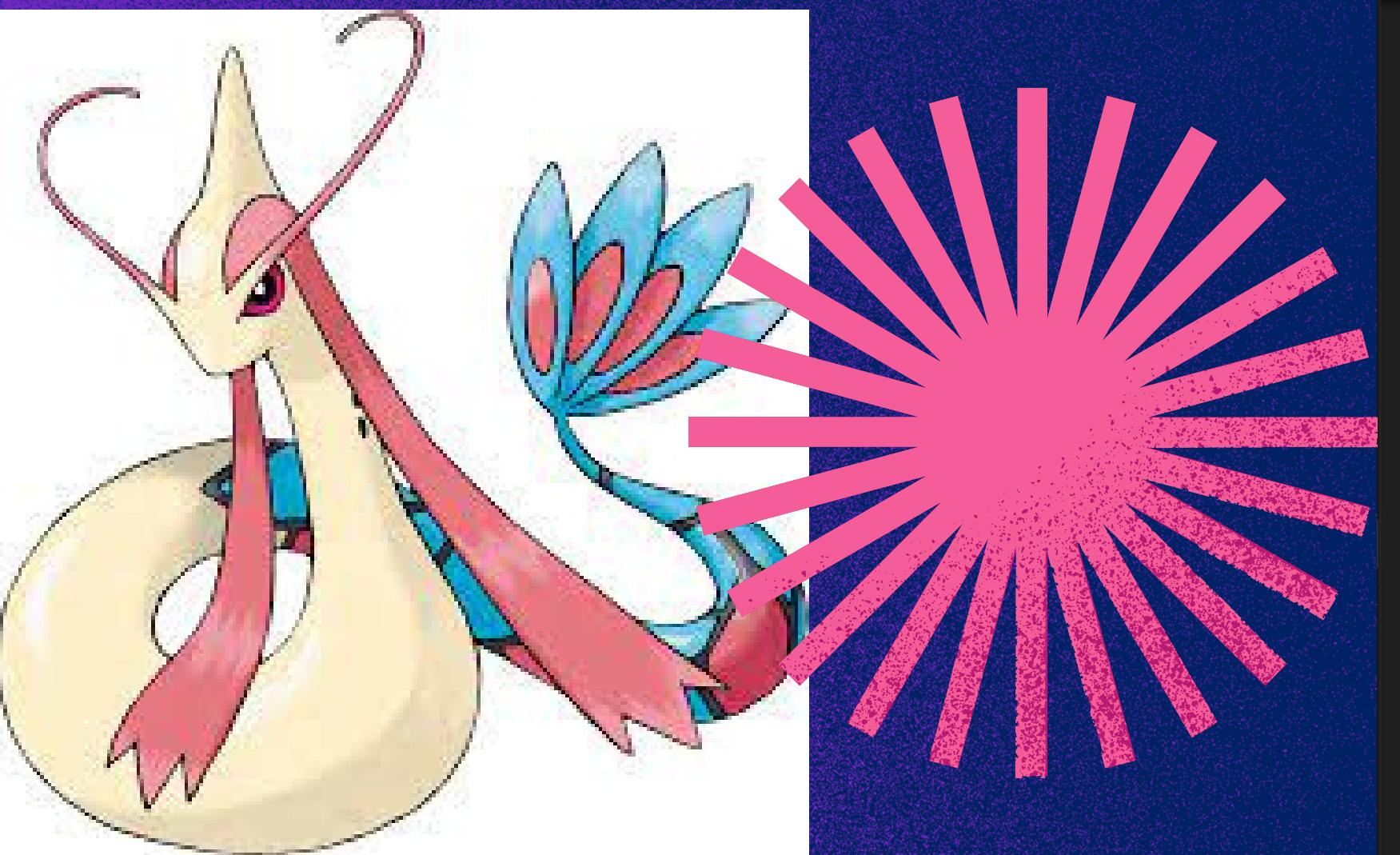
```
pokedexgui.py X
```

```
pokedexgui.py > ...
```

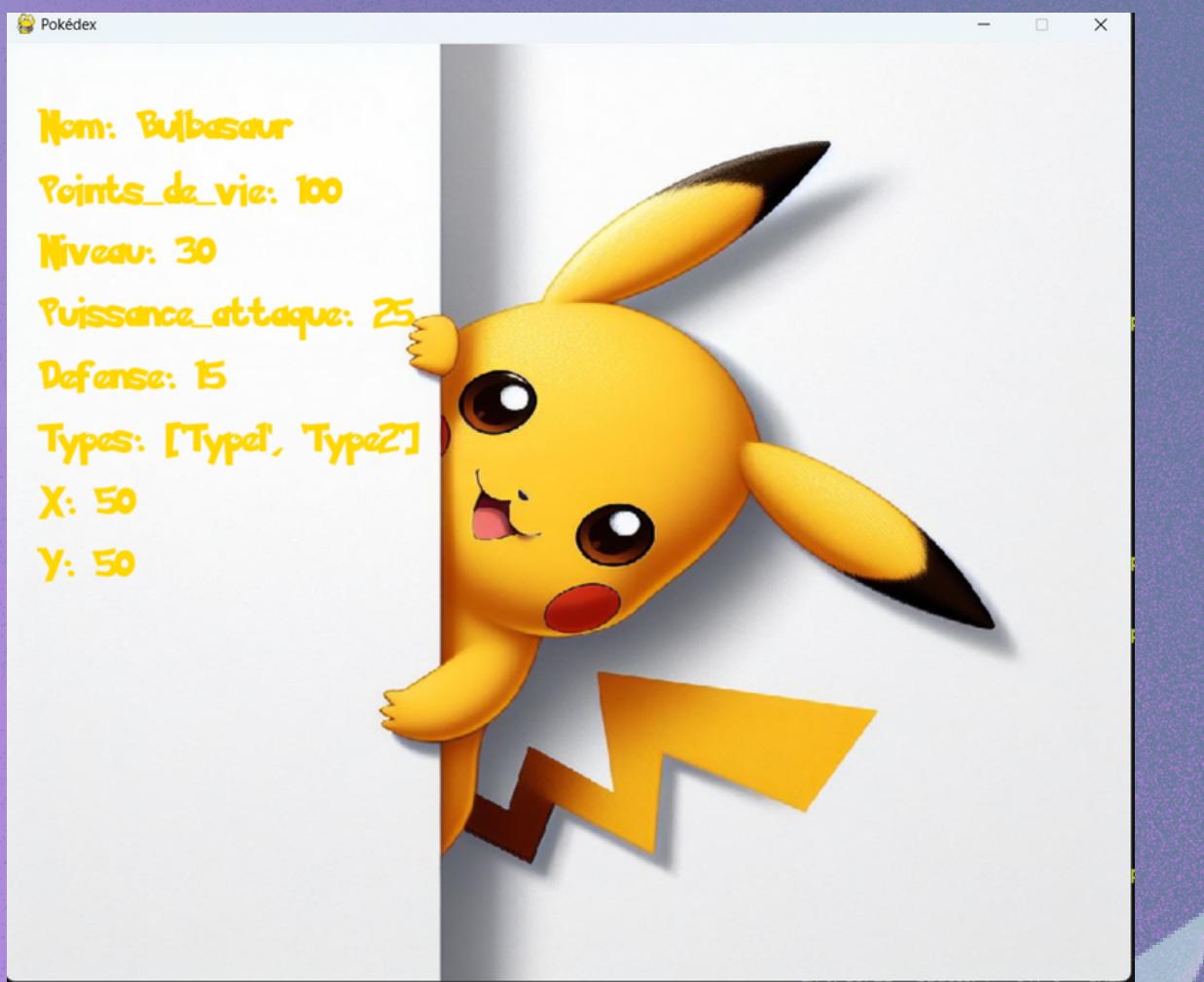
```
You, 1 hour ago | 1 author (You)
```

```
1 import pygame
2 import json
3
4 pygame.init()
5
6
7 with open("pokedex.json", "r") as file:
8     pokemon_data = json.load(file)
9
10
11 width, height = 800, 600
12 screen = pygame.display.set_mode((width, height))
13 pygame.display.set_caption("Pokédex")
14
15
16 background = pygame.image.load("Imagepokemon/Pokedex.jpg")
17 background = pygame.transform.scale(background, (width, height))
18
19 # Couleurs
20 white = (255, 255, 255)
21
22 # Police
23
```

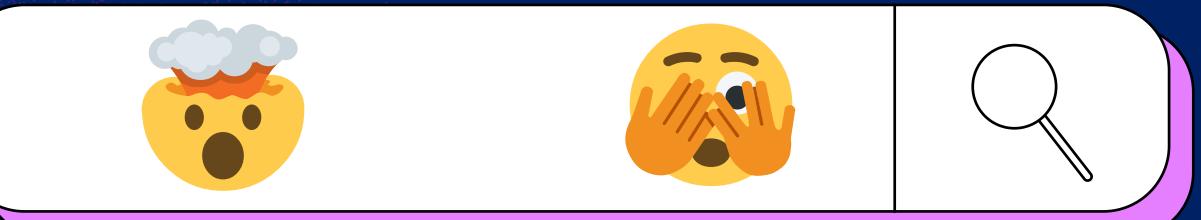
# Une partie du code du pokedex.py



```
1 import json
2 You, 4 weeks ago • pokedex
3
4 You, 3 weeks ago | 1 author (You)
5 class Pokedex:
6     def __init__(self, save:str=None):
7         self.data = [None] * 24
8         if save is None:
9             with open("pokedex.json", 'r', encoding='utf-8') as f:
10                 pokdata = json.load(f)
11                 for k in range(18, 24):
12                     self.data[k] = pokdata[k]
13                 self.save_dex()
14         else:
15             with open(save, 'r', encoding='utf-8') as f:
16                 self.data = json.load(f)
17
18     def read_dex(self):
19         data = []
20         for k in range(len(self.data)):
21             if self.data[k] is not None:
22                 data.append(self.data[k])
23
24         return data
```

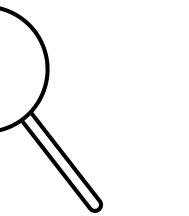


# CLASS POKÉDEX :

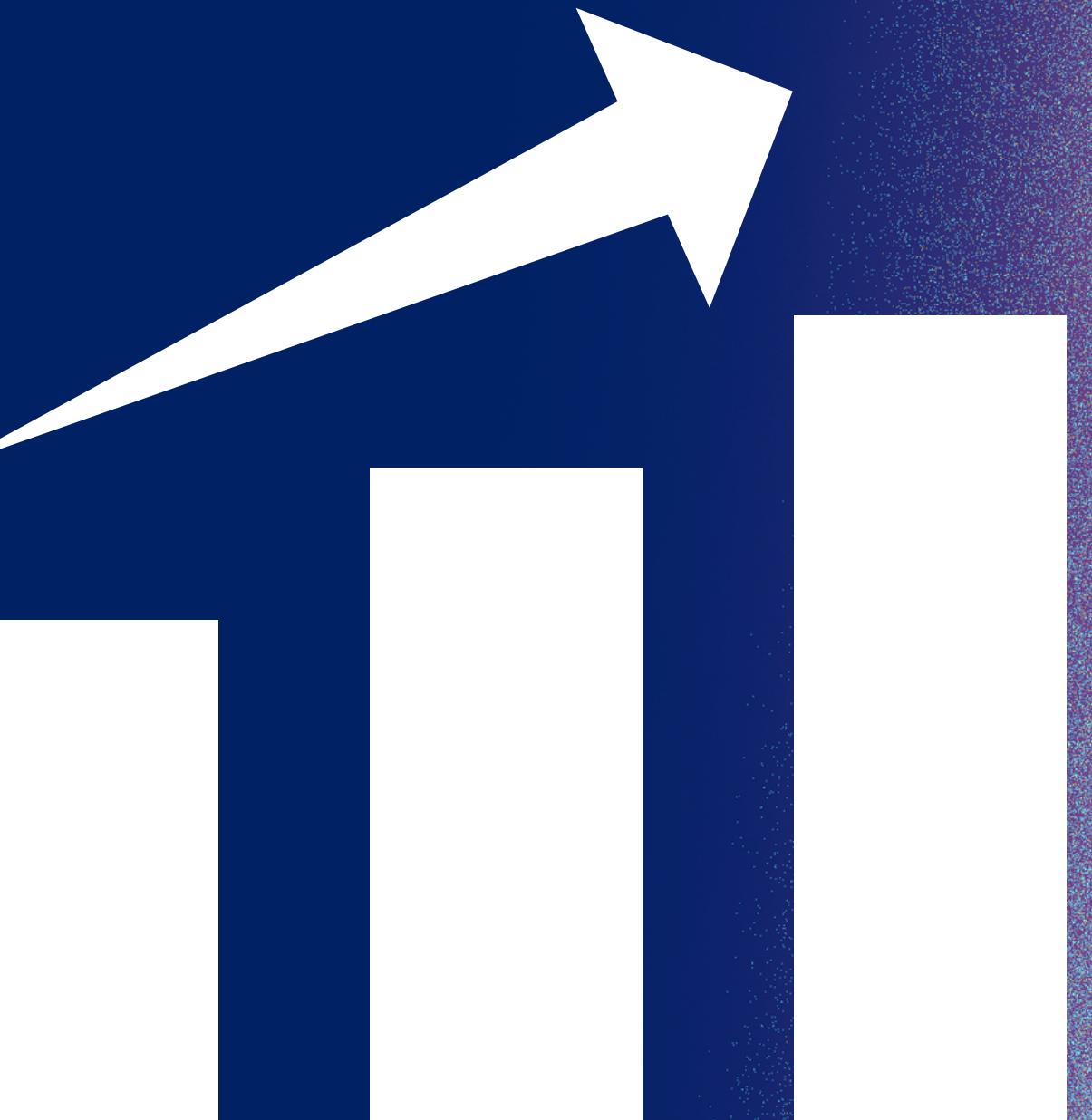


Finalement on à ce résultat.

# Progression et Améliorations dans notre Projet



- **Compréhension approfondie de Python**
- **Familiarisation avec Pygame**
- **Création et interconnexion de classes**
- **Amélioration de la création d'interfaces utilisateur**
- **Travail en équipe renforcé**





- Communication et Collaboration
- Planification et Gestion du Temps
- Structure du Projet.
- des difficultés pour interconnecter toutes les classes et les branches entre elles.
- La gestion de Git, en particulier avec l'opération de merge.

# MERCI À VOUS!



Kamelia, Maysa, Sofia

