Projet Pokémon

Python



https://github.com/cbarange/pypoke-game

Lancer le jeu depuis le fichier */src/UIPyPoke_V1.py*

```
pip install pygame
pip install requests
pip install Pillow
```

Table des matières

I. Présentation du sujet

Afin de terminer et de valider le module de Python, nous devions réaliser un jeu ayant pour thème principal « Pokémon ». L'objectif de ce projet est de créer un jeu permettant :

- De contrôler un personnage devant capturer des Pokémons
- La capture ne doit pas être automatique
- Les captures doivent être enregistrées (Pokédex)
- Obtenir des informations sur ces Pokémons
- Créer des équipes (4-5) et les faires combattre

II. Choix des outils

Nous avons choisi de d'utiliser une approche standalone. Ainsi l'application cliente devra être indépendante, nous avons cherché une librairie python capable de produire des jeux standalone. Nous sommes rapidement tombés sur la librairie PyGame qui bénéficie d'une grande communauté. Celle-ci permettait de gérer de manière simple l'affichage et la collision entres les éléments du jeu.

Nous avons fait le choix d'un affichage en 2 Dimensions, facilitant l'affichage et le déplacement du personnage. Des images contenants l'ensemble des éléments graphiques sont accessibles en local. Le programme va alors chercher un élément précis d'une image avant de l'afficher à l'utilisateur.

Au vu des nombreuses fonctionnalités demandées nous avons choisi d'en prioriser. Comme l'affichage du jeu de manière esthétique avec des sprites plutôt qu'un affichage console standard.

III. Segmentation des taches

Pour faciliter le développement nous avons choisi de segmenter les tâches. Cela nous a permis d'avancer sur de nouvelles fonctionnalités sans implémenter la totalité des précédentes. Voici les différents stades que nous avons définis :

- Première version :
  - Fenêtre d'une dimension fixe de 768 x 480
  - Déplacement du personnage avec 4 sprites
  - Affichage d'éléments de décors
  - Le déplacement dans les hautes herbes peut déclencher un combat
  - Fuir le combat et revenir sur la fenêtre principale
- Deuxième version :
  - Combattre les pokémons adverses
  - Afficher les informations du personnage
  - Capturer les pokemons adverses
- Troisième version :
  - Changer de zone en allant sur le bord de l'écran
  - Combattre les autres dresseurs en allant dessus
  - Gagner des récompenses
  - Gestion de l'inventaire
- Quatrième version :
  - Les desseurs adverses se déplace sur la carte
  - Rentrer dans les bâtiments
  - Ajout de son et de musique
  - Déplacement avec 16 sprites
- Cinquième version :
  - Gestion de l'utilisation d'une manette
  - Gestion du multi-joueurs

IV. Ressources

L'ensemble des éléments graphiques proviennent des images contenues dans le répertoire *Images*.

Figure 1 - Images utilisés pour l'affichage.

Les données associées aux pokemons sont récupérées depuis l'api PokéAPI (https://pokeapi.co/). Nous utilisons une structure de données JSON pour stocker les pokemons. Cette structure est encapsulée dans une classe python *PokemonBO*.

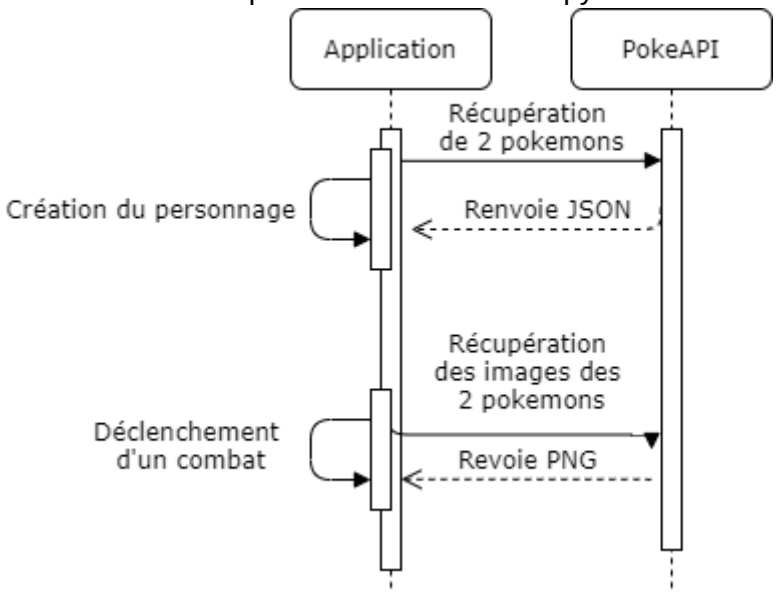

Figure 2 - Diagramme de séquence de l'utilisation général de l'API.

```
'name': 'pikachu',
'id': 25,
'level': 7,
'pv': 29,
'fullPV': 29,
'type':'normal',
'attacks': [
      {'name': 'captivate', 'power': None, 'type':'normal'},
      {'name': 'brick-break', 'power': 75, 'type':'normal'},
      {'name': 'dynamic-punch', 'power': 100, 'type':'normal'},
      {'name': 'rollout', 'power': 30, 'type':'normal'}],
'stat': {'def': 40, 'att': 55, 'basePV': 35},
'xp': 0,
'nextLevel': 50,
'capture_rate': 190,
'evolution_chain': 'https://pokeapi.co/api/v2/evolution-chain/10/',
'image': {
    'front': 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/25.png',
    'back': 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/25.png'
    },
'flavor_text': '群れて\u3000暮らす\u3000森は\n落雷が\u3000絶えず\u3000危険だ。',
'captureEnable': True
```

Figure 3 - Exemple de la structure de données pour pikachu.

Le processus de création de cette structure se passe en plusieurs temps, en effet en fonction de la zone où se trouve le joueur les pokémons rencontrés sont différents. Les échanges entre l'api et notre jeu sont assez importants et parfois le temps de chargement d'un combat est visible avec un écran noir. C'est le temps que toutes les requêtes reçoivent une réponse.

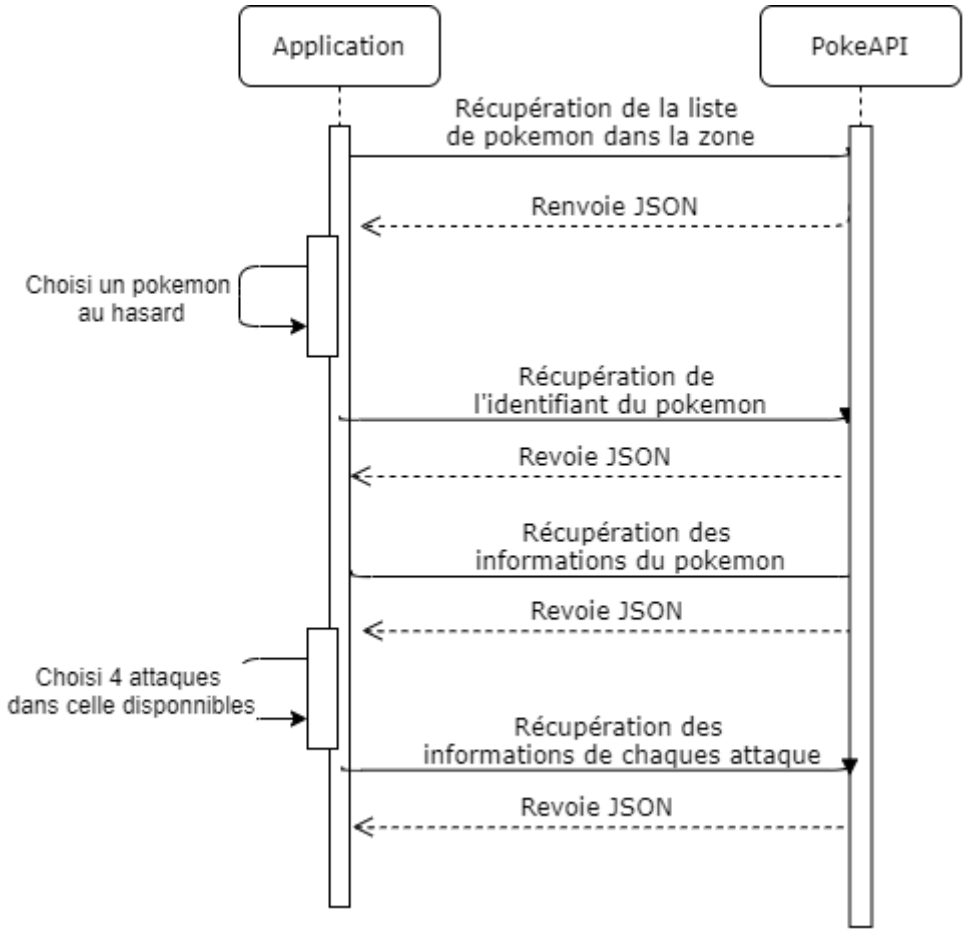

Figure 4 - Processus de création d'un pokémon.

Nous utilisons un total de 4 adresses différentes :

- Récupération des pokémons accessibles dans la zone
  - https://pokeapi.co/api/v2/pal-park-area/*ID*/
- Récupération des informations globales du pokémon
  - https://pokeapi.co/api/v2/pokemon-species/*ID*/
- Récupération des informations détaillées du pokémon
  - https://pokeapi.co/api/v2/pokemon/*ID*/
- Récupération des informations des attaques
  - https://pokeapi.co/api/v2/move/*ID*/

V. Paramètres

Bon nombre de paramètres du jeu sont facilement modifiables depuis le fichier *constants.py*.* Nous avons choisi de rassembler les paramètres du jeu dans un unique fichier pour faciliter la modification, notamment pour les images et les touches de claviers.

```
"""Constantes du jeu PyPoke"""

# Paramètres de la fenêtre
windows_size = (768, 480)
windows_title = 'PyPoke [EPSI-Python]'
icone_image = '../Images/ico.jpg'
fps_max = 60

# Paramètres du jeu
walk_speed = 3
run_speed = 6
combat_chance = 20 # On a une chance sur combat_chance de declancher un combat
display_delay = 3

# Paramètres de l'affichage
perso_size = 32
perso_size_tuple = (perso_size,perso_size)
perso_scale = (128,128)
hit_box = False

# Zone en fonction du plateau
areaType = {1:'mountain',2:'field',3:'pond',4:'field',5:'field',6:'field',7:'foret',8:'field',9:'sea'}

# Listes des images du jeu
accueil_image = '../Images/background.jpg'
fond_image = '../Images/background.jpg'
fondNoir_image = '../Images/fond_noir.jpg'
fondgrass_image = '../Images/backgroundGrass.png'
[ .... ]

# Touches, /!\ Pygame est en qwerty /!\
move_right = K_d # Deplacement vers la droite
move_left = K_a # Deplacement vers la gauche
move_bottom = K_s # Deplacement vers le bas
move_top = K_w # Deplacement vers de haut
courir = K_LALT # Permet de courrir
interact = K_e # Interaction avec un element
hit_box_show = K_h # Non fonctionnel
fuire_combat = K_SPACE # Retour a l'ecran principal
info_key = K_i # Affichage des information du personnage
```

Figure 5 - Constantes utilisées dans le jeu, permet de changer rapidement un paramètre.

VI. Fonctionnement du jeu

Le jeu est assez simple, le joueur doit battre ou capturer des pokémons sans que son équipe ne meure en entier. Pour cela il commence avec un pikachu ainsi qu'un autre pokémon prit au hasard dans la zone *field*. Il commence sur le plateau 8. La carte compte un total de 9 zones avec chaque une leur difficulté.

1 FORET	2 POND	3 POND
4 FORET	5 FORET	6 FIELD
7 MOUNTAIN	8 FIELD	9 SEA

Figure 7 - Répartition des plateaux avec les zones.

```
# 8 --> 5-10
# 5,9 --> 10-20
# 1,2,3,6 --> 15-25
# 4 --> 20-30
```

Figure 6 - Répartition des niveaux des pokémons en fonction des plateaux.

Si le joueur gagne un combat il gagne des *Poke$*. Pour le moment ce gain ne sert pas à grand-chose mais on peut imaginer qu'ils serviront à acheter des bonus que le joueur pourra utiliser en combat contre un autre joueur.

Pour déclencher un combat il suffit de marcher dans les hautes herbes, et il y a une sur 20 pour qu'un combat démarre. Ceci est paramétrable dans le fichier *constants.py*. Le pokémon sauvage est alors choisi en fonction des pokémons disponibles dans la zone. Durant un combat l'ensemble des actions sont affichées sur la gauche. Cela permet au joueur d'avoir quelques informations, comme l'attaque de l'adversaire ou encore s'il a réussi à capturer le pokémon. Pour réussir une capture il faut avoir au moins un score de 1. Nous avons choisi de rester au plus près des statistiques proposées par le jeu original. Nous nous sommes inspirées des formules du jeu, (https://bulbapedia.bulbagarden.net/wiki/Damage).

Figure 9 - Plateau 8, zone de départ.

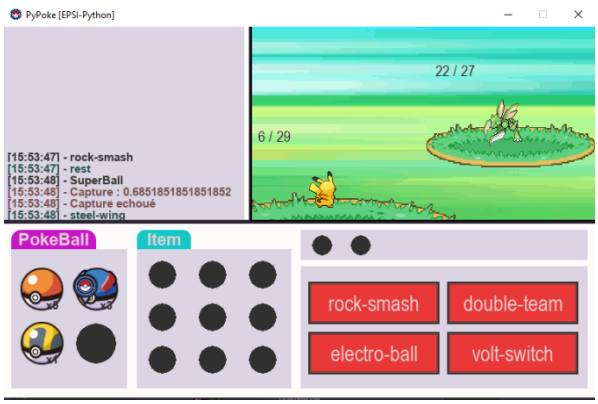

Figure 8 - Déroulement d'un combat.

Le joueur est en permanence informé des changements divers au cours du jeu. Un système de message popup permet d'adresser des messages personnalisables.

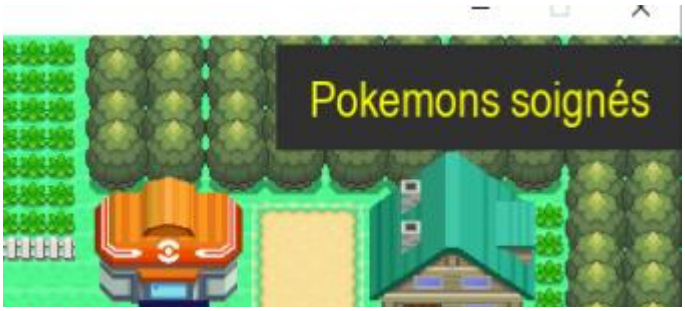

Figure 10 - Exemple du message popup.

Nous avons mis en place un système de combat qui prend en compte les différents aspects des pokémons ainsi que le temps sur le plateau.

```
def weatherCoefficient(typePokemon,weather):
    """
    --- Power effect by weather status ---
    sunny   → nothing
    rain    → fire↓25% water↑25% grass↑15%
    snow    → fire↓15% water↓15% grass↓15% rock↓15% ice↑25%
    thunder → electric↑25%
    """
```

Figure 12 - Répartition des bonus en fonction du temps sur le plateau.

Figure 11 - Répartition des bonus d'attaques en fonction du type du pokémon adverse.

VII. Conclusion

En conclusion le projet était intéressant le libre choix du framework nous a permis d'expérimenter la librairie *PyGame*, cette librairie est très facile à utiliser dans un premier temps mais il faut vite revoir la conception « traditionnelle » d'une application. En effet *PyGame* aborde une approche game-dev avec la gestion d'une boucle d'affichage. Pour ne pas interrompre cette boucle nous avons utilisé des variables d'états applicatifs ainsi qu'un certain nombre de variables globales. *PyGame* est une librairie complète mais qui ne dispose pas d'élément complexe. En effet pour faire un rectangle avec des angles courbé il faut utiliser tous un ensemble de stratagèmes.

Nous n'avons malheureusement pas été jusqu'au bout du développement, nous avons validé jusqu'à la troisième version du plan que nous avions établi. D'autres projets ont démarré au même moment et nous avons mis un peu de temps avant de maitriser complètement les paradigmes du *game development.*