**RAPPORT PROJET POKEMON**

Organisation du code

. Nous avons scindé le code en 6 parties principales (6 fichiers python). La première s’intitulant **“Pokemon”** qui permet dans un premier temps de récupérer les informations sur les pokémons utilisés, comme les points de vie, à l’aide du fichier csv “pokemon\_first\_gen”, puis dans un second temps de créer un dictionnaire où pour chacun de ces pokémons on a leurs caractéristiques, et enfin de voir si ce pokémon est sauvage ou au dresseur.

. La seconde partie, **“player\_turn”** permet de créer l'interface de combat ainsi que les mécaniques principales liées à l’interface graphique lorsque c’est le tour du joueur : la possibilité de victoire, de défaite, et les différents boutons cliquables, que sont l’attaque neutre, l’attaque spéciale et la fuite. Cette partie permet également de faire jouer l’adversaire, et définie les conditions de victoire et de défaite.

. La troisième partie, nommée **“combat\_code”**, utilise les deux premières. Elle utilise la première via les attributs récupérés pour chaque pokémon, pour définir les dégâts pris par ces derniers en fonction de leur type (faiblesse/résistance) via un coefficient multiplicateur appliqué aux attaques spéciale et neutre de la deuxième partie. Ceci s’applique au pokémon sauvage comme à celui du dresseur, et cette troisième permet également l’actualisation de l’interface graphique, notamment des pv des deux pokémons. Enfin, la fonction lancement\_combat affiche et ferme la fenêtre de combat.

. La quatrième partie, **“ClassesMapDresseur”** sert à créer deux classes, Map et Dresseur. Tout d’abord est récupéré le tableau des noms des pokémons et de leurs coordonnées à l’aide du fichier csv “pokemon\_coordinates” qui est réutilisé dans la classe Map. Cette classe permet de calculer la distance entre le dresseur et les pokémons sauvages, ce qui permettra par la suite de savoir s'il faut afficher le pokémon ou non. La classe Dresseur permet d’obtenir le pokémon de départ utilisé par le joueur dans une liste qui grossira au fil des captures du dresseur.

. La cinquième partie s’appelant **“main”** est le cœur du jeu permettant de faire fonctionner les quatre parties précédentes ensemble et de faire tourner le jeu en temps réel. Premièrement, on y charge les images de la carte, des pokémons à afficher, du dresseur, et du bouton de combat. Ensuite, on définit deux distances arbitraires, une à partir de laquelle on affichera le pokémon, et une autre (plus courte) à partir de laquelle on affichera le bouton permettant de lancer le combat. Dans la boucle du jeu, on affiche d’abord la carte, ainsi que le dresseur, et on implémente les mouvements en fonction de la touche utilisée. Pour afficher les pokémons, on utilise la distance définie arbitrairement et la classe Map de la partie quatre. Même chose pour le bouton de combat. Ensuite, si le joueur presse le bouton de combat, le combat se lance en utilisant la partie trois avec la fonction lancement\_combat. On implémente avant le début du combat la possibilité pour le joueur de choisir le pokémon qu’il veut utiliser pour combattre selon ceux se trouvant dans sa liste définie dans la classe Dresseur de la partie quatre, au début il n’y en a qu’un. Dans le cas de défaite et de fuite, il ne se passe rien, la fenêtre de combat se ferme et le joueur retourne sur la map. En cas de victoire (si les pv du pokémon sauvage tombent à zéro) alors le pokémon est ajouté à la liste des pokémons possédés par le dresseur et il est effacé de la map. Enfin il y deux conditions à l’arrêt du jeu : soit le joueur presse la touche ECHAP, soit il a capturé les 151 pokémons présents sur la carte.

. Enfin, la sixième partie, **“Menu”,** permet comme son nom l’indique d’initialiser un menu avant que le jeu commence, permettant de cliquer sur “nouvelle partie” ou quitter, appuyer sur nouvelle partie lance le code se trouvant dans “main”, et appuyer sur “quitter” ferme simplement la fenêtre.

Choix faits (adaptation/compromis)

. Les coordonnées fournies dans l’énoncé sont celles utilisées (elles sont décimales), de fait il a fallu adapter des coordonnées à la taille de la map pour que les pokémons rentrent tous dedans, et ils sont donc à des positions bien précises, parfois sur une colline ou sur un arbre, ce qui ne fait pas forcément sens, mais ce choix d’avoir gardé les coordonnées de l’énoncé pour les 151 pokémons a fait que nous n’avons pas pu implémenter de collisions avec le décor (hormis les bords de la map). Ces coordonnées impliquent aussi une casi superposition de certains pokémons sur la map, mais encore une fois, au vu de l’échelle de la carte, c’est un problème non solvable si on garde les coordonnées de l’énoncé.

. Dans le “main”, l’utilisation de pygame est privilégiée à celle de Qt Designer pour sa simplicité à afficher les fenêtres, les images, et à implémenter les déplacements du dresseur.

. Le problème lors de l’affichage du pokémon avec pygame, est qu’il n’est pas possible de le “désafficher” simplement après l’avoir vaincu, nous avons essayé dans un premier temps de passer la portion de la map se trouvant sous le pokémon au premier plan pour “faire disparaître” le pokémon mais comme les coordonnées ont été modifiées pour correspondre à l’échelle de la carte, cette méthode était trop fastidieuse. Nous avons donc simplement rendu transparents le pokémon et son bouton de combat associé après la victoire, ce qui techniquement n’empêche pas le joueur de cliquer à nouveau sur le bouton pour relancer le combat.

. La taille de la fenêtre de combat n’est pas la même que celle de la fenêtre. (Choix)

Bugs encore présents/ non corrigés :

. Dans le “main” : Lorsque le combat se ferme après la défaite, la victoire ou la fuite, la map ne répond plus. Le problème vient sûrement d’un conflit entre la fenêtre de combat réalisée avec Qt et la fenêtre de la map réalisée avec pygame. On a essayé plusieurs choses, comme ouvrir la fenêtre de combat Qt via pygame dans le “main”, ou encore de lancer les deux fenêtres séparément, quitte à ce qu’il soit encore possible de se déplacer sur la map alors que le combat est en cours, mais rien n’y fait, le bug persiste. La seule option aurait été de refaire la fenêtre de la carte/pokémons/dresseur avec Qt mais le temps était insuffisant.

. Dans “Menu” : nous avons le même problème avec le menu, c’est pour ça qu’on l’a séparé du “main” et qu’on le fait tourner à part. Avec le menu, la map ne répond plus avant même qu’on lance un combat. La raison est sûrement la même que précedemment, un conflit entre la fenêtre du menu réalisé avec Qt et celle de la map, ou alors simplement que la superposition de fenêtres est mal gérée par pygame.