Mario

ABOU JAMRA

TG3

**NSI-Carnet de bord projet pokemonMMS**

1. Cahier des charges
2. Contexte et présentation du projet :

Développement d’un jeu Pokémon en python. Là où traditionnellement les jeux Pokémon ont une partie exploration et une partie combat, nous avons décidé de réaliser un jeu Pokémon avec uniquement une partie combat.

Le joueur joue donc contre un bot (PVE), celui qui gagne le combat gagne la partie. Pour l’instant nous comptons réaliser un jeu fonctionnel avec des fonctions basiques, pour peut-être rendre le jeu plus complexe plus tard.

La phase de jeu se décompose en plusieurs étapes après le lancement :

* Le joueur choisit de jouer dans un menu
* Le joueur sélectionne ses Pokémons un par un.
* Le combat commence, jusqu'à ce que le joueur ou le BOT n’aient plus de Pokémons qui puissent combattre (un Pokémon ne peut plus combattre quand ses PV sont égaux ou inférieurs à 0).

Durant la phase de combat le joueur peut réaliser différentes actions :

* Attaquer ou utiliser une capacité non-offensive
* Soigner son Pokémon grâce à des potions (leur nombre est prédéfinis avant le combat et semblable pour le BOT et le joueur)
* Changer de Pokémon actif

Le combat se déroule en tour par tour, le premier qui attaque étant celui qui a le Pokémon le plus rapide. Le joueur interagit avec le jeu via l’interface graphique en cliquant ou il souhaite effectuer une action.

Après une victoire ou une défaite, le joueur se voit afficher un message et la possibilité de rejouer.

Nous souhaitons également réaliser un système où le joueur rentre son pseudo avant de jouer afin de conserver ses statistiques. Cela signifie créer une interface supplémentaire afin de pouvoir rentrer son pseudo et un autre pour consulter les statistiques.

1. Besoins et contraintes liés au projet :

Nous avons prévu que notre projet se sépare en trois codes pour des choix de coordinations. De plus nous mettrons en place un Drive afin de travailler à distance dans les meilleures conditions. La première page de code concerne l’interface graphique et appellera les fonctions de la deuxième page de code qui concerne toutes les autres fonctions. La troisième concerne les statistiques.

Interface graphique :

Pour l’instant nous avons prévu de réaliser sept interfaces différentes :

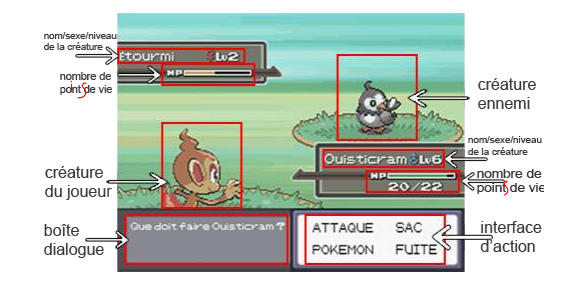
* Menu (choix de jouer ou accès aux statistiques)

Si choix de jouer :

* Saisi du pseudo du joueur
* Choix du premier Pokémon parmi ceux disponibles
* Choix du second Pokémon parmi ceux disponibles
* Choix du dernier Pokémon parmi ceux disponibles
* Introduction du combat (par exemple : un homme nous interpelle pour nous racketter notre goûter 🡪 combat)
* Interface du combat
* Interface de fin de combat (« Bravo », etc.)(accès aux stats ou rejouer)

Si accès aux statistiques :

* affichage des statistiques par pseudo (ratio victoire/défaite)

Interface supposée de combat : 

Programmation autre que l’interface graphique :

La deuxième partie du code concerne donc toutes les fonctions autres que l’interface graphique, pour en citer le maximum, les attaques et les capacités qui ont chacune leurs effets, les fonctions qui initialise le nombre de potions en début de partie, les fonctions qui initialise les équipes de Pokémons, etc. Notre code tourne autour d’une classe principale, la classe Pokémon qui permet d’initialiser un Pokémon comme tel.

Classe et attributs de la classe :

* Classe Pokémon :
  + Nom (nom du Pokémon) (Str)
  + TypePokemon (chaque Pokémon à un type qui influe sur les autres Pokémons) (Str)
  + PV (nombre de point de vie du Pokémon) (Integer)
  + Vitesse (nombre de point de vitesse du Pokémon) (Integer)
  + Attaque (nombre de point d’attaque du Pokémon) (Integer)
  + Esquive (nombre de point d’esquive du Pokémon) (Integer)
  + Défense (nombre de point de défense du Pokémon) (Integer)
  + Attaque1 (Capacité offensive) (Str)
  + Attaque2 (Capacité offensive) (Str)
  + Attaque3 (Capacité offensive) (Str)
  + Capacité (Capacité non-offensive (effets passifs)) (Str)
  + Type\_cap (impact de la capacité non-offensive (exemple : defense\_down)) (Str)

Méthodes de la classe Pokémon :

* \_\_init\_\_ (self, nom, typePokemon, pv, vitesse, attaque, esquive, defense, attaque1, attaque2, attaque3, capacite, type\_cap)
  + Initialise un Pokémon, plus la variable pvBase qui est égale aux PV du Pokémon (self.pvBase = pv)
* NomAttaque (self, adversaire)
  + Une attaque est une capacité qui inflige des dégâts physiques. La quantité de dégâts infligés dépend de nombreux facteurs, de la puissance d’attaque de l’attaquant, de son type par rapport au Pokémon qui subit (un type eau aura l’avantage sur un type feu, ses dégâts seront multipliés par exemple), la défense du Pokémon qui subit (une défense élevée réduit les dégâts infligés), de plus la vitesse du Pokémon offensif peut octroyer un bonus de dégâts sous certaines conditions, tout comme la vitesse du Pokémon qui défend peut lui octroyer un bonus défensif. Le Pokémon qui subit peut tenter d’esquiver si l’attaque peut être évitée (en fonction de ses points d’esquive). Retourne l’efficacité de l’attaque et met à jour les paramètres du Pokémon (PV).
* NomCapacité(self) ou (self, adversaire)
  + Une capacité non-offensive influe sur les statistiques du Pokémon cible. Un Pokémon peut donc augmenter son attaque via une capacité ou baisser la défense de son ennemi, etc. Retourne une chaîne str qui nous indique l’effet appliqué. (Met à jour les statistiques ciblées)

Ici nous avons décidé de ne pas faire une fonction attaque en général qui nous éviterait de rédiger plein de fonctions, nous avons choisis pour des questions de simplicité dans le code de rédiger chaque attaque une à une. De plus ce sera utile sur le long terme si on veut rajouter des Pokémons, les capacités et attaques pouvant être communes.

* nomSoins (pokemon, sac)
  + Prend en paramètre le sac (une liste initialisée en début de partie (voir plus bas)), et le Pokémon qui doit recevoir les soins, une fois une potion utilisée elle disparaît. Il existe trois positions différentes, la « potion » qui rajoute 20 PV, la « super-potion » qui en rajoute 50, et « l’hyper-potion » qui en rajoute 100. Un Pokémon ne peut pas se régénérer au-dessus de ses Pv de base. Retourne une chaîne str pour nous informer que le soin a été effectué.
* queFaire (self, adversaire, equipeEnnemie, sacEnnemie)
  + Méthode qui permet de définir ce que le bot doit faire après qu’on ait joué. Il y a trois actions possibles, si son Pokémon est K.O, il devra effectuer un changement obligatoire pour continuer à combattre. Si son Pokémon n’a plus beaucoup de vie, il aura la possibilité d’utiliser une potion adaptée. Enfin, s’il n’a plus la bonne potion ou qu’il n’a pas besoin de se soigner, le BOT attaquera ou utiliseras une capacité en faisant appel à la méthode attaqueBot qui va définir la meilleure à utiliser.
* attaqueBot (self, adversaire)
  + Méthode qui permet de déterminer la meilleure attaque ou capacité à utiliser pour le Bot. La méthode fait un appel à cette attaque ou cette capacité afin de l’appliquer.

Les méthodes de la classe Pokémon prennent en compte de nombreux éléments extérieurs à cette classe comme par exemple « equipeEnnemie » ou encore « sacEnnemie ». Ces variables seront initialisées en début de partie grâce aux fonctions ci-dessous.

Fonctions extérieures à la classe Pokémon :

* Equipe (equipe) : (liste(str))
  + Fonction qui initialise les deux équipes en fonction des choix faits sur l’interface graphique. La liste « equipe » contient les noms des Pokémons choisis, puis la fonction les transforme en objet de la classe Pokémon.
  + Selon les Pokémons choisis, la fonction initialise l’équipe ennemie.
  + Elle nous retourne la liste « equipe » et la liste « equipeEnnemie » qui contient les Pokémons des équipes respectives.
* Sac () :
  + Fonction appelée en début de partie, crée et retourne deux listes :
    - sacEnnemie, composé de 3 potions tel qu’un objet d’indice i égal une potion.
    - sac, composé de 3 potions tel qu’un objet d’indice i égal une potion.

Malo : Code global (initialisation de partie, BOT, attaques)

Mario : Interface graphique (design des différentes interfaces, interactions joueurs-interface)

Sylvain : Réalisation d’un système de statistiques et de pseudo , création de l’interface graphique liée aux statistiques

1. Déroulement du projet :

Inspirée du célèbre jeu Pokémon, le projet pokemonMMS est une version simplifier du jeu original où seul la face combat est présente. Lors des premiers jours de travaux sur le projet, nous avons dû réfléchir sur la répartition des tâches (trouvable sur le cahier des charges) ainsi que sur quel module développer le projet. En comparant les différents modules (pygame,tkinter…) nous avons opter pour pygame, c’est donc à partir de ce moment où le vrai travail commence.

Avant de commencer ma partie (interfaceMMS et animation) j’ai dû me familiariser avec l’utilisation de pygame, il s’est donc passé de nombreuses heures sur l’apprentissage de pygame. Etant donné que ma partie était l’une des plus importante j’ai dû commencer à rapidement faire des tests.

La première fenêtre (fenêtre principale) mis énormément de temps à se faire, en effet, la documentation en ligne de pygame (<https://www.pygame.org/docs/>) fut l’un des documents le plus utilisé à la réalisation de ma partie, elle m’a non seulement permis d’apprendre et de connaitre l’utilisation des fonctions propre à pygame mais elle m’a notamment aidez à corriger facilement les erreurs rencontré, grâce au exemple de code accessible sur le site de la documentation en ligne.

A la suite de cette même fenêtre, j’ai pu crée plusieurs fenêtres où il est possible de choisir les pokémons qui composeront notre équipe lors du combat. Evidemment ces même fenêtre sont en adéquation avec le codeMMS où se situe la partie principale du fonctionnement (définir l’équipe etc..). En réalité dire que la définission l’équipe se passe dans cette partie du code n’est pas exact. En effet je n’ai que simplement mis en relation à travers de cette interface graphique le joueur avec la partie du codeMMS.

Au bout d’un certain temps le groupe du projet a atteint un certain rythme et une certaine routine où premièrement Malo faisait sa partie du code dans codeMMS où réalisait ensuite des tests. En parallèle je travailler sur la création d’une interface permettant d’interagir avec ce code et une fois les deux parties terminer nous mettions en commun nos deux travaillent. La partie la plus dure fut la mise en commun, en effet la plupart du temps consacrée aux séances du projet étaient utilisé à la mise en commun de nos deux codes.

A noté que c’est pendant le confinement que notre projet a réellement commencer à prendre forme, avec l’ajout des pokémons dans l’interface de combat et leur animation qui donne une certaine vivacité au jeu le rendant plus mouvementer et agréable à jouer. Mais le plus simple étaient derrière nous car c’est à ce moment-là que les choses sérieuses ont commencé avec la mise en commun des attaques et des autres actions possibles, car de nombreux problèmes furent rencontrer.

1. Problème rencontré :

Avant de passé aux problèmes lié aux codes en lui-même, je vais expliquer l’une des actions liées à la réalisation du projet qui fut une **Grande** perte de temps. C’était l’animation des pokémons et des images/format utilisé. Avant tout test, j’avais prévu d’utiliser le format .gif pour faire une animation correcte et non encombrante mais ce format n’était pas supporté par pygame (une simple image était affichée au lieu du gif en entier) c’est pourquoi j’ai dû me renseigner sur les animations pygame et c’est là que j’ai appris à animer quelques choses (objet, animal, tout…) à l’aide d’une succession d’image mis rapidement à la suite. Renommé les images, les importer/redimensionner et les exploiter était long et mettait un réel désordre au code interfaceMMS (qui devenait de plus en plus long) c’est pourquoi l’animation liée aux pokémons fut coupé d’interfaceMMS et déplacée vers un nouveau code « animation ».

Une fois le nouveau code parfaitement fonctionnel avec l’aide de Malo, je mettais en accord les deux principaux codes, mais au niveau graphique certaine image s’affichait avant ou après l’endroit voulue c’est pourquoi nous avons donc mis le code dans un certains ordre d’exécutions pour qu’ils puissent afficher correctement et au bon moment les images.

Après les images correctement mises, nous avions réussi a obtenir quelque chose d’a peu près jouable (le combat se déroulait mais notre pokémon de prenait pas de dégâts). Le bouton pokemon fut donc l’un des prochains objectifs visés par notre groupe. En effet j’avait rencontrer de nombreux problème et étant donné que Malo avait pratiquement terminer le codeMMS il a pu m’aider à la création de « Pokemon » (le bouton qui permet de changer de pokemon). C’était l’une des parties les plus longues d’interfaceMMS (au niveau de la réalisation).

Le reste (bouton Sac et bar de pv) ne fut pas réellement un problème car Sac fonctionner à peu près de la même manière que Pokémon.

1. Explication partie interfaceMMS et animation :

Le code interfaceMMS commence par l’importation des autres codes (codeMMS,animation) suivit de l’initialisation des variables utilisées dans le code et des fonctions utilisées. Chaque fenêtre est définie par un numéro avec la variable « nfenetre » donc à chaque changement de fenêtre, il suffisait de mettre à jour la variable selon la fenêtre voulue.

Après avoir choisit le Pokémon nous croisons l’ennemie du jeu qui nous attaque et lance donc le combat contre nous. Le joueur a quatre choix possibles :

- utiliser son sac pour soigner un Pokémon

- utiliser Pokémon pour changer de Pokémon

- attaquer le Pokémon ennemie

- fuir (qui n’est pas vraiment une option réalisable)

Chaque fois que le joueur faisait une action, le Bot exécutait son tour avec la fonction Quefaire de codeMMS. Au fil de la partie la personne n’ayant plus de Pokémon pouvant se battre (K.O) perdait. C’est un fois le combat terminer qu’intervient la partie de Sylvain, ou les statistiques Score,victoire,défaire… peuvent être afficher avec comme mémoire un blanc note où sont stocker ces variables.

Conclusion :

Ce projet et ce travaille m’a appris le travail en groupe, de plus en ces périodes de confinement où il est compliqué de rassembler le groupe du projet nous avons dû nous organiser et réorganiser sur le développement du projet (nous avons d’ailleurs de nombreuses fois changé nos affections de partie). Comme par exemple la mise en place d’un drive commun qui fut très utiles car chaque personne mettait leurs codes chaque fois qu’il avait travaillé dessus. Nous avons aussi utilisé des moyens de communications, en effet vers les dernières semaines du projet nous avions pris du retard et donc pour palier à ce problème nous mettions en place nos mises en commun via des partages d’écran. L’aide du groupe était une très bonne chose car ils nous permettaient d’avoir un second point de vue ainsi qu’un aide à la réalisation de certaine partie compliquer.