Rapport Partie 2

Thomas Varin, Arthur Adjedj

March 2022

1 État du projet

Le projet est fonctionnel et contient les fonctionnalités attendues pour les parties 1 et 2, et plus encore.

2 Approche au développement

Comme lors de la partie 1, nous avons subdivisé nos tâches afin de travailler de manière quasi-isolée sur nos travaux: là où Thomas s'est concentré sur la construction d'une carte et d'un système de déplacement, ainsi que l'amélioration de systèmes déjà présents comme la boite de dialogue, Arthur s'est concentré sur la construction du pokédex, ainsi que de la partie graphique du jeu. Nos travaux avaient tendance à se chevaucher, Thomas a notamment du travailler sur l'ajout de boutons et de sliders dans la pokédex, alors qu'Arthur a optimisé l'affichage des boutons et le système de grille de la map. Finalement, une communication efficace entre les acteurs du projet a permis la bonne organisation de celui-ci.

3 Choix et difficultés rencontrées

3.1 Types

Les types étant déjà présents lors de la première partie, on se contente ici de redécrire leur fonctionnement:
Les pokémons et attaques possèdent tous un type parmi les 6 suivants : Normal, Feu, Eau, Plante, Électrique et Glace. Lorsqu'une attaque de type x est appliquée à un pokémon de type y, un multiplicateur est appliqué à cette dernière suivant cette grille:

x / y	Normal	Feu	Eau	Plante	Électrique	Glace
Normal	1	1	1	1	1	1
Feu	1	1/2	1/2	2	1	2
Eau	1	2	1/2	1/2	1	1
Plante	1	1/2	2	1/2	1	1
Électrique	1	1	2	1/2	1/2	1
Glace	1	1/2	1/2	2	1	1/2

Aussi, si une attaque a un coefficient multiplicateur fort ou faible face à un certain pokémon, l'utilisateur en est notifié par la boîte de dialogue :

3.2 Objets

Comme pour les types, les objets ont déjà été implémentés dans la partie 1 et sont pleinement fonctionnels dans les combats. On retrouve aussi des objets sur la carte qui ont un intérêt sur cette dernière, comme un vélo pour aller plus vite, une clé pour ouvrir une porte et une planche de surf pour se déplacer sur l'eau.

3.3 IA

En ce qui concerne l'intelligence artificielle, nous avions le choix entre un système d'arbre de recherche et un système "fait main". Nous avons opté pour la seconde option, avec l'idée qu'un parcours d'arbre n'était pas très intéressant, en terme de temps de calcul mais surtout en terme d'intérêt pédagogique.
C'est pour cela que nous avons opté pour un système pensé pour être efficace, simple à utiliser et contenant une part d'aléatoire. Pour cela, nous avons

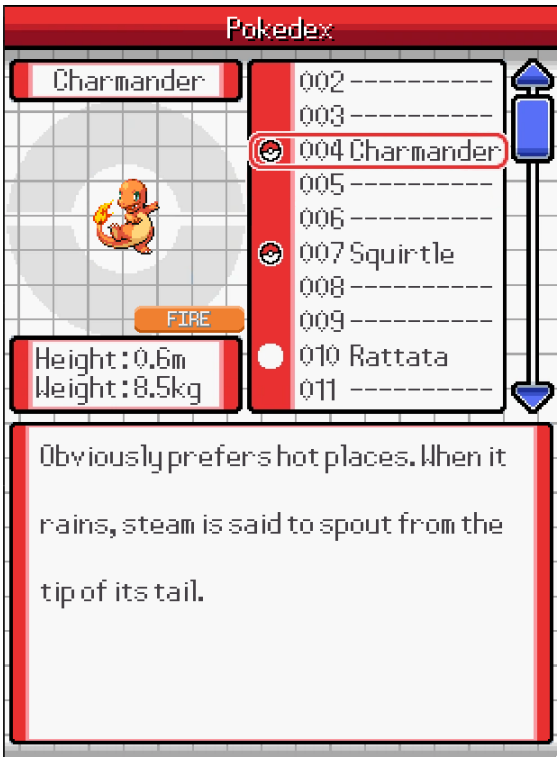
implémenté les traits ScoreForStrategy et Intelligence. Le premier est un trait que l'on donne aux actions réalisables par l'IA (attaque, buff, application d'un statut ou encore changement de pokémon), le second à l'IA elle même. Le premier trait impose une fonction de score, qui évalue l'utilité d'une action.
Celle-ci se base sur des coefficients que nous avons arbitrairement choisi mais prend en compte les bonnes variables (une attaque qui fait plus de dégâts est meilleure, un pokémon avec un meilleur type contre l'adversaire sera meilleur...).
Une fois cela fait, on réuni toutes les actions dans une liste et on en choisi une au hasard avec une probabilité pondérée par les scores. Cette pondération s'appuie notamment sur la stratégie donnée au joueur, qui n'est rien d'autre qu'un ensemble de 5 entiers.
Alors, on peut aisément attribuer une stratégie offensive en privilégiant les attaques, comme on peut préférer une stratégie défensive en privilégiant les buffs/debuffs.

3.4 Carte

Pour la carte, nous disposons d'une grille de blocs (de taille arbitraire) qui contient des listes de blocs, comme des blocs empilés, pour une question d'affichage (comme par exemple une pierre sur de la glace). Si rien n'est spécifié, la liste est simplement composée d'un EmptyBlock, l'équivalent de l'air.
Le personnage peut alors s'y déplacer librement, tant que les blocs sur lesquels il veut aller le permettent. En effet, certains blocs ne peuvent pas être traversés, certains seulement sous certaines conditions (l'eau ne peut être traversée qu'avec un surf par exemple).
Ce système nous permet alors de construire très facilement de nouvelles cartes, nous en avons deux auxquelles nous pouvons accéder en utilisant des portails (nous projetons d'y accéder avec les bordures des cartes).
Pour ces cartes, nous avons construit un certain nombre de blocs différents. Nous avons déjà parlé de l'eau et des portails, mais on peut aussi évoquer les portes qui s'ouvrent et se ferment en interagissant avec, et qui peuvent se déverrouiller à l'aide de clés que l'on trouve sur la carte ou en battant des adversaires. La simplicité de notre système le rend très facile d'utilisation, ce qui permet de créer des nouveaux blocs très facilement, comme de la glace qui nous fait glisser.
Tous les personnages sont animés : ils marchent pour se déplacer et ont une orientation. On peut interagir avec eux pour les combattre si ceux-ci n'ont pas déjà été battus. En plus de cela, certains ont la capacité de repérer le joueur afin de l'intercepter et de combattre contre lui.
Enfin, nous avons ajouté un système de caméra dynamique. Le joueur se déplace dans une sorte de boîte de laquelle il ne peut sortir. Si celui-ci tente de sortir de ce cadre, c'est toute la carte qui le suit. Ce mouvement de caméra rend le jeu beaucoup plus fluide.

3.5 Pokédex

Le pokédex est un outil complet permettant d'en apprendre beaucoup sur les pokémons rencontrés lors de votre aventure ! Lorsqu'un pokémon est rencontré pour la première fois, lui ainsi que toute les informations qui lui sont associées apparaissent dans le pokédex : son numéro, son nom, sa taille, son poids, son type, ainsi qu'une courte description de ce dernier.
Il est possible de déplacer dans le pokédex de diverses manières, le rendant très interactif Ainsi, ce système de navigation complet permet d'accéder de manière intuitive aux informations nécessaires. Si un pokémon n'a pas encore été rencontré, son nom ne sera pas affiché, et ses informations ainsi que son image seront cachés. Aussi, pour différencier les pokémons rencontrés des pokémons attrapés, un petit symbole de pokéball apparaît à coté des noms des pokémons attrapés, alors qu'un cercle vide est présent à la place pour les pokémons simplement rencontrés mais pas attrapés, voir exemple ci-contre :



3.6 Améliorations diverses

En plus des multiples ajouts cités précédemment, de multiples améliorations aux systèmes déjà produits lors de la première partie du projet on été implémentées, en voici une liste non-exhaustive:
-Ajout de 54 nouveaux pokémons, il ne manque que 5 pokémons pour compléter

le roster souhaité.
-Boite de dialogue améliorée: le texte apparaît maintenant de manière progressive,un buffer stocke l'ensemble des textes à afficher dans la boite de dialogue. Il est possible d'accélérer l'apparition du texte, en cliquant par exemple sur sa souris ou sur entrée. Un seul bloc de texte pour la boite de dialogue peut maintenant être constitué de plus de 2 lignes. Si tel est le cas, le texte descend progressivement pour afficher les lignes 2 par 2 lors de l'apparition du texte.
-Le système de boutons est maintenant plus complet. Les design de certains boutons a été revu. Les types des attaques et des pokémons sont maintenant affichés sur leurs boutons respectifs.
-Dans un soucis d'optimisation, un système de mémoïsation d'images a été mis en place. En effet, pour la carte par exemple, il est possible qu'une même image soit utilisée pour différents blocs (un rocher peut par exemple apparaître plusieurs fois à l'écran). Pour éviter de recharger la même image pour chacun de ces blocs, les $BufferedImage$ chargées lors de l'exécution du programme sont gardées en mémoire et recyclées dès que nécessaire.

3.7 Quelques difficultés

Au cours de cette deuxième partie, nous avons rencontré quelques difficultés.
La première, et la plus importante, vient du langage lui-même. En effet, scala lance au démarrage des threads pour son fonctionnement. En revanche, un seul thread gère l'affichage et les événements, ce qui ne nous permet pas de faire attendre le thread d'événement pour faire une animation (puisque l'affichage ne se fera tout simplement pas). C'est pourquoi il est nécessaire d'instancier des threads pour gérer toutes les animations, et d'être très consciencieux lorsque l'on travaille avec les événements.
En plus de cela, scala ne nous permet pas l'utilisation de variable statique ou de fonction statique. C'est pourquoi nous avons tant de variables dans notre objet Utils. Cependant, nous avons pris connaissance du système de compagnon, qui nous permettra de résoudre ce problème pour la troisième partie.
Aussi, nous n'avons pas été en mesure de faire tous les assets graphiques à temps, notamment pour certains objets comme le vélo et le surfboard. Ceux-ci seront rajoutés pour la partie 3.