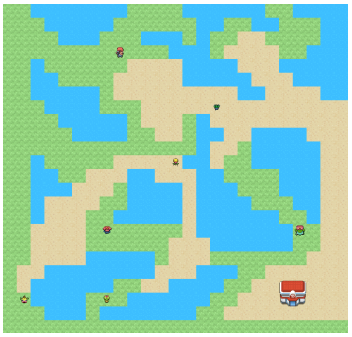
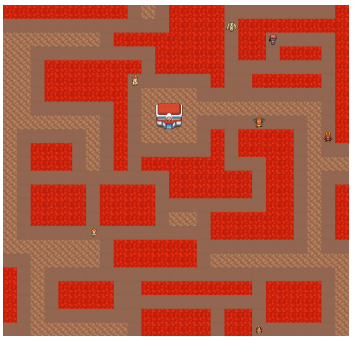
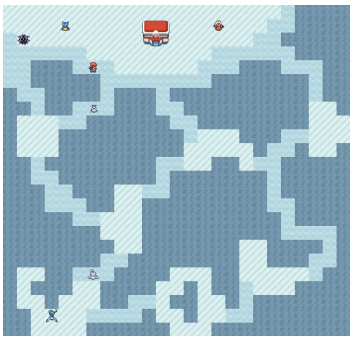
Introduction :

Notre jeu reprend les bases de Pokémon : le joueur explore une Map, découvre des Pokémons, participe à des combats dont les types des Pokémons font varier les dégâts subis, et il constitue une équipe de Pokémons.

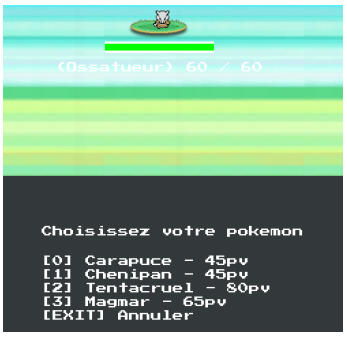
Démonstration :

En lançant le jeu, le joueur découvre les règles de celui-ci et la façon de jouer. Il pourra ensuite choisir son personnage et son Starter. Alors il arrivera sur une Map où des Pokémons sont présents. Il y a 6 Maps en tout, dont celle du Boss, qui sont en concordance avec les types des Pokémons présents.

Il peut se déplacer et interagir avec un Pokémon d'eux pour le combattre.

Lors du combat, le joueur peut choisir le Pokémon qui attaquera l'ennemi parmi son inventaire. Il pourra alors choisir entre deux attaques : **une attaque normale**, et une **attaque en fonction de son type**. Les dégâts sont calculés à partir des types des Pokémons, ainsi que leur puissance d'attaque initiale. L'attaque normale (Charge) est le produit de la puissance d'attaque initiale du Pokémon par 0.25. Par exemple, pour Bulbizarre, l'attaque normale vaut 12, et **plus le Pokémon est puissant, plus son attaque normale sera élevée**. Quant à l'attaque typée, il s'agit du produit de cette attaque normale par un coefficient de dégâts qui prend en compte les types du Pokémon attaqué et de l'attaquant. Par exemple, si les deux Pokémons ont le même type, ce coefficient vaut 0.5, tandis que si l'attaquant est de type Eau et l'attaqué de type Feu, ce coefficient vaut 2.



S'il perd, il pourra aller soigner son Pokémon au centre de soin et choisir un nouveau Pokémon à combattre.

S'il gagne, le Pokémon ennemi rejoindra son inventaire, que le joueur peut consulter à tout moment. Il atterrit alors sur la map suivante, et ce jusqu'au boss.

Lorsque le joueur arrivera au niveau final, il n'aura d'autre choix que de se battre contre Mewtwo. S'il perd, il débloque automatiquement Mew dans son inventaire à la place de son Starter, ce qui lui permettra de vaincre Mewtwo plus facilement. Si Mewtwo est vaincu, le joueur aura gagné et fini le jeu.

Utilisation :

Avant de jouer :
[D], [F], [G], [H], [J] : choisir son personnage.
[0], [1], [2] : choisir son Starter.

Sur la Map :
[Z], [Q], [S], [D] ou **Flèches**: se déplacer sur la carte.
[M] : changer de vue sur la carte.
[I] : ouvrir ou fermer l'inventaire.
[E] : interagir avec un Pokémon, ou soigner ses Pokémons au centre de soin.

Pendant le combat :
Echap : sortir du combat .
[0] à [6] : choisir le Pokémon qui attaquera.
[E] et [G] : choisir son attaque.

Implémentations :

Notre projet a été réalisé à l'aide de la bibliothèque **SDL**. Elle nous a permis d'implémenter une interface graphique, animée, composée d'images et de textes (avec ses extensions *SDL_Image* et *SDL_TTF*). Nous avons dans un premier temps suivi une série de tutoriels de Carl Brich sur la technologie SDL. A partir de l'épisode 7, nous avons dû nous émanciper de ses tutoriels puisqu'ils abordent des fonctionnalités un peu trop avancées.

Nous avons décidé de mener notre projet à l'aide de la **programmation orientée objet**. L'intérêt de cette technologie est multiple. D'une part, elle permet la création d'architectures complètes tout en bénéficiant des avantages de l'encapsulation, de la composition et des héritages. Nous avons notamment utilisé le principe des classes abstraites (et même des interfaces) pour la classe **Interface.hpp**. Elle est parente de toutes les interfaces créées: *InventoryInterface.cpp*, *ExplorationInterface.cpp*, etc...

Afin d'éviter les redondances de condition, nous avons utilisé un système de vecteur dans le Game.cpp, à l'appel des différentes interfaces.

```
20      std::vector<Interface *> interfaces;
```

```
63          interfaces.push_back(starterInterface);
64          interfaces.push_back(explorationInterface);
65          interfaces.push_back(attackInterface);
66          interfaces.push_back(inventoryInterface);
67          interfaces.push_back(endingInterface);
```

```
125 /**
126  * @brief Refresh the game
127  */
128 void Game::refresh() {
129     for (size_t i = 0; i < interfaces.size(); i++) {
130         if (interfaces[i]->isActive()) {
131             interfaces[i]->handleEvents();
132             interfaces[i]->update();
133             interfaces[i]->render();
134         }
135     }
136 }
137
```

```
1   Bulbizarre 45  49  49  1439 0  1
2   Carapuce   45  49  49  1727 0  2
3   Salameche  45  49  49  1439 64 3
4   Triopikeur 35  100 60  287  64 4
5   Mystherbe  45  50  55  383  0  1
6   Rafflesia  75  80  85  479  0  1
7   Boustiflor 65  90  50  767  0  1
8   Chetiflor  50  75  35  671  0  1
9   Florizarre 80  82  83  1535 0  1
10  Chenipan   45  30  35  1823 0  1
```

Enfin, afin de répertorier les pokemons plus efficacement, nous avons mis en place une sorte de base de données. (database/pokemon.txt)

Cette base de données est appelée dans le constructeur de la classe Pokemon.

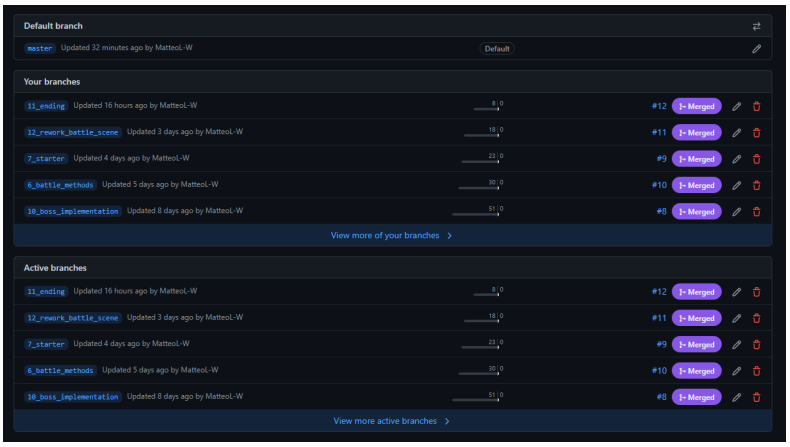
Difficultés :

C'était notre première utilisation de la technologie SDL ainsi que de la POO sur le langage C++. Bien sûr, nos choix de conceptions nous ont posé quelques soucis : notamment sur la gestion de la mémoire. Nous avons eu affaire à plusieurs fuites de mémoires, globalement toutes résolues. Nous avons également été dans l'incapacité d'installer un potentiel debugger, nous avons donc fait sans, durant l'intégralité du projet.

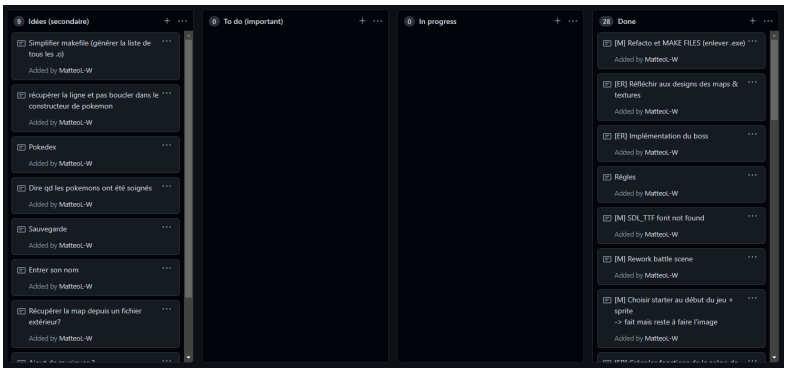
Gestion de projet :

Le projet a été réalisé à l'aide de Git et est actuellement en open-source sur GitHub.

Nous avons utilisé le système de **branches** et de pull-requests de git



Et le système de **Kanban** mis en place sur la plateforme



Merci de votre lecture !

Emily-Rose Strich - Mattéo Leclercq