

Rapport:
Projet PI4 2022:
Fire Emblem sur le thème de Pokémons

AMARA Mouloud
GHANEM Wissam
LE NINIVEN Adrien
TOKO Nawfel
TRAN Henri Bergson

Mai 2022
GitLab du projet

Table des matières

1	Introduction	1
1.1	Présentation	1
1.2	Objectif du projet	1
2	Cahier des charges	1
2.1	Conception du terrain	2
2.2	Conception de l'UI	3
2.3	Règles du jeu	4
2.3.1	Déroulement de la partie	4
2.3.2	Rareté des Pokémons	4
2.4	Problèmes rencontrés	5
2.4.1	Algorithmique	5
2.4.2	Java Swing	5
2.5	Pistes d'extensions	6
3	Rôle(s) de chacun et difficultés rencontrées	6
3.1	GHANEM Wissam et TOKO Nawfel	6
3.2	AMARA Mouloud	7
3.3	LE NINIVEN Adrien	9
3.4	TRAN Henri Bergson	10
4	Conclusion	11
5	Diagrammes des classes	11

1 Introduction

1.1 Présentation

Notre Projet est une adaptation sur ordinateur en langage JAVA du jeu Pokéémon Conquest, lui-même inspiré de la célèbre série de jeux vidéo Fire Emblem.

1.2 Objectif du projet

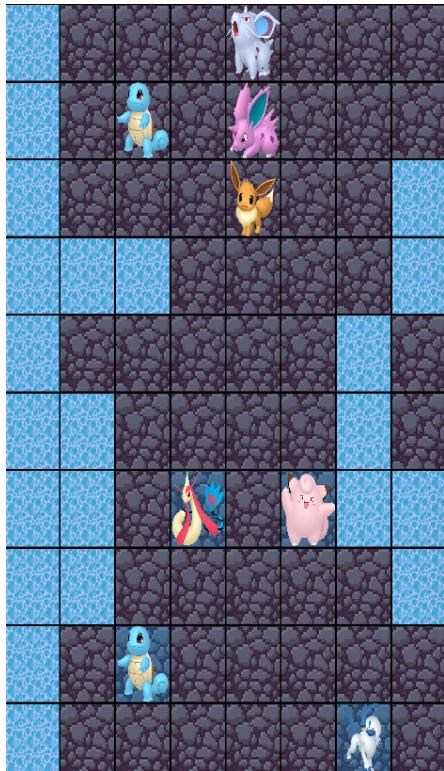
Notre projet consiste en l'implémentation de l'ensemble des éléments de base que possède le jeu Pokéémon Conquest à travers un modèle VMC (View Model Controller) composé de plusieurs classes JAVA, afin de modéliser les différents aspects du jeu pour ainsi pouvoir jouer une/plusieurs partie(s) à partir d'un interface graphique claire et optimisée.

2 Cahier des charges

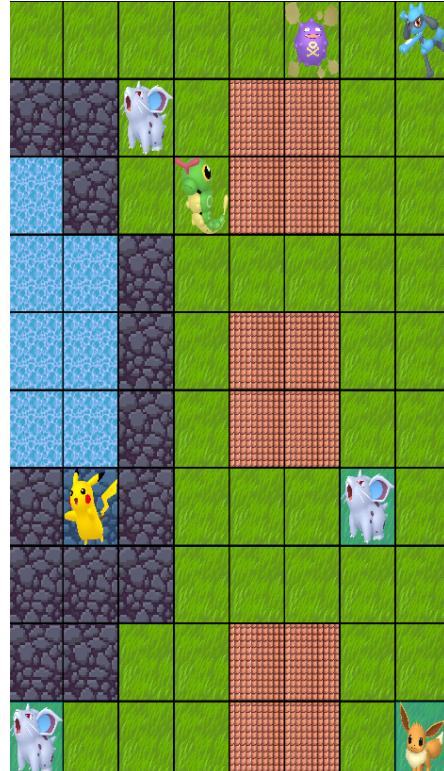
- Mise en place d'un plateau avec différents types de terrain.
- Les Pokémons ont des points de vie et des attaques variées, de portée différentes.
- Lors du déplacement d'un personnage, les cases accessibles sont coloriées en bleu.
- On peut voir quels Pokémons ont déjà joué.
- Système de tour par tour. Fin de partie lorsqu'un joueur n'a plus de Pokémons.
- affichage complet des différentes informations en temps réel.
- Pouvoir recommencer la partie et avoir des terrains et Pokémons choisis au hasard.
- Affichage de l'historique du jeux.
- Colorier en rouge les cases des Pokémons que l'on peut attaquer.
- Pouvoir annuler un déplacement.
- Les attaques peuvent engendrer des effets différents sur les Pokémons selon leur type.

2.1 Conception du terrain

La plupart des jeux Fire Emblem utilisent en guise de terrain de jeu une grille 2D avec des cases correspondant à différents types de terrain sur lesquelles certains Pokémons peuvent se déplacer selon leur type. Nous avons donc naturellement repris ces mêmes principes et en même décider d'utiliser la bibliothèque java Swing pour l'ensemble de la partie interface graphique du projet. Pour ce qui est de l'aspect technique du terrain, il s'agit d'un JPanel qui possède plusieurs Tiles également de type JPanel, organisées avec une mise en page de type GridLayout.



(a) Montagne

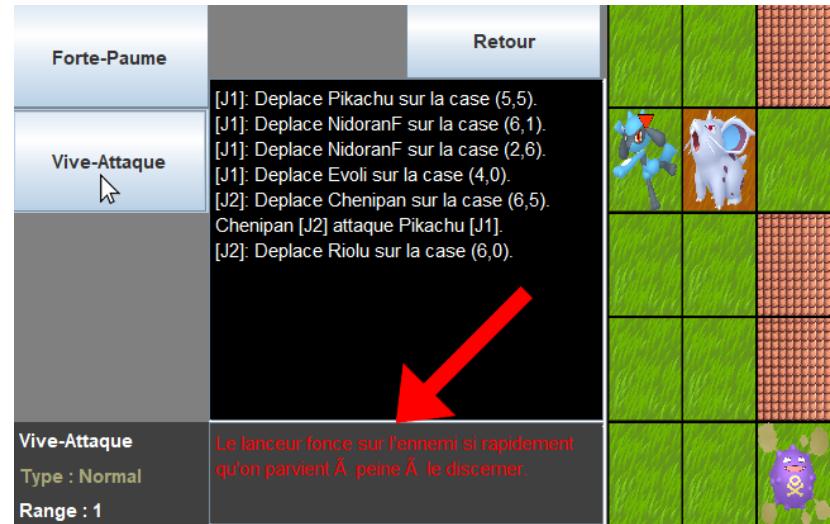
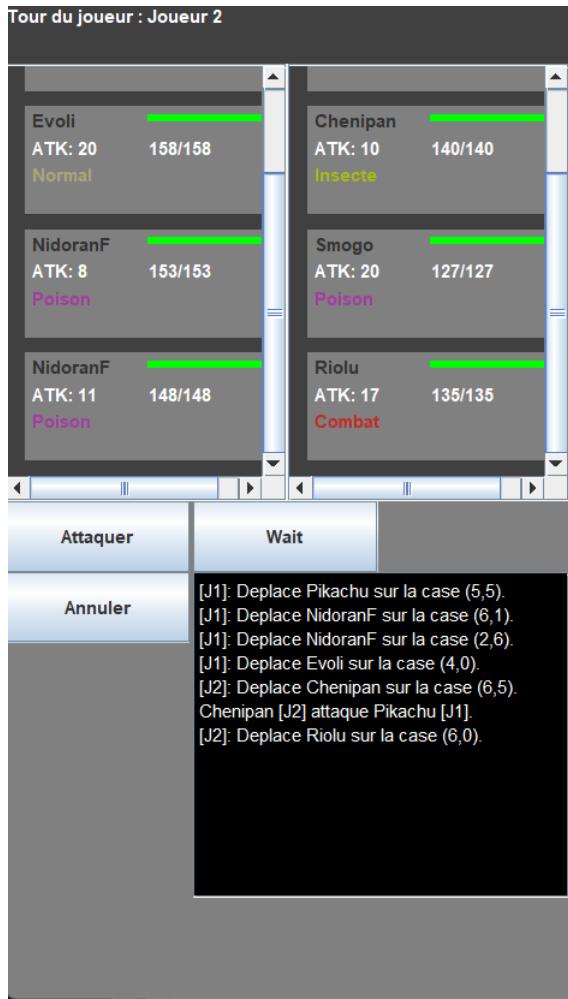


(b) Village

FIGURE 1 – Exemples de terrain

2.2 Conception de l'UI

Un des axes les plus importants à traiter lors du développement d'un jeu vidéo est la communication entre le joueur et le jeu. La difficulté ici est de montrer un maximum d'informations à l'écran du joueur de façon claire et ordonnée afin que celui-ci puisse prendre les décisions rapidement, mais surtout pour qu'il développe sa stratégie. Le gameplay est également très important car il faut que le joueur puisse réaliser ses coups facilement grâce à des boutons et divers repères visuels. Nous avons également implémenté une musique ainsi que des effets sonores afin de rappeler l'ambiance des combats Pokémons et surtout transmettre d'avantages d'informations aux joueurs. Typiquement, lorsqu'un joueur déclenche une attaque sur un Pokéémon, celle-ci générera un effet sonore différent selon son efficacité.



(b) Panneau affichant les informations à propos de l'attaque que le curseur pointe

(a) Interface global (informations à propos des Pokémons, boutons d'action et panneau des logs)

FIGURE 2

2.3 Règles du jeu

2.3.1 Déroulement de la partie

Lorsque l'on clique sur le bouton "Jouer", deux fenêtres correspondant aux Joueur 1 (à gauche) et Joueur 2 (à droite) apparaissent côte à côté et ainsi débute le premier tour. Chaque joueur se voit attribuer 4 Pokémon aléatoirement. Sur la fenêtre du Joueur 1, les cases où se trouvent ses Pokémon apparaissent en bleu ce qui signifie qu'il peut les sélectionner pour les déplacer. Une fois un de ses Pokémon déplacé, le joueur peut choisir d'attaquer seulement si il est suffisamment proche d'un Pokémon ennemi pour qu'il puisse l'attaquer, et donc selon la portée de l'attaque. Dans le cas où l'attaque est possible, les cases du ou des Pokémon ennemi(s) se trouvant dans la portée de l'attaque apparaissent en rouge. Lorsque le joueur clique sur une de ces cases, un son retentit et l'on peut voir les points de vie infligés au Pokémon grâce aux logs, mais aussi à la barre de vie de celui-ci. Si le joueur ne peut attaquer aucun Pokémon, il doit cliquer sur le bouton "Wait" pour qu'il puisse déplacer un de ses autres Pokémon. A la fin de son attaque, un Pokémon ne peut plus agir avant le prochain tour du joueur auquel il appartient. Une fois que le Joueur 1 n'a plus aucun coup à réaliser, le jeu passe au tour du joueur 2 et suit donc les mêmes règles citées précédemment. La partie se termine lorsque l'un des deux joueurs réussit à mettre K.O. l'ensemble des Pokémon de son adversaire. Pour relancer une partie, les joueurs ont le choix d'appuyer sur le bouton "Recommencer" pour initialiser une nouvelle partie.

2.3.2 Rareté des Pokémon

Comme la sélection des Pokémon est aléatoire, nous avons décidé d'instaurer un système de rareté. Étant donné que certains Pokémon ont de meilleures statistiques que d'autres, il y a donc moins de chances de les obtenir.

Voici les taux de rareté :

- **Communs** : 55%
- **Peu communs** : 35%
- **Rares** : 10%

Rareté	Pokémon
Communs	Abo Abra Chenipan Smogo Evoli Rattata NidoranF NidoranM Melofee Voltobre
Peu com- muns	Salamèche Bulbizarre Carapuce Pikachu Riolu Minidraco Riolu Nodulithe
Rares	Absol Oniglali Ronflex Milobelus

2.4 Problèmes rencontrés

2.4.1 Algorithmique

Au moment où nous voulions implémenter les déplacements des Pokémons, nous nous sommes retrouvés face à notre premier casse-tête algorithmique. En effet plusieurs paramètres sont à prendre en compte afin qu'un Pokémon puisse se déplacer sur une certaine case et pas sur une autre, comme le type de case, ou si un Pokémon est déjà présent etc...). Notre professeur nous a donc redirigé vers une implémentation dans un tableau à deux dimensions de l'algorithme BFS (*Breadth-First Search*) permettant ainsi de stocker dans une unique liste l'ensemble des cases où le Pokémon peut se déplacer. Il nous a donc fallu faire quelques retouches au niveau des variables afin d'adapter l'algorithme à notre projet.

2.4.2 Java Swing

Au départ nous avons choisis la bibliothèque java swing car nous étions assez à l'aise avec son fonctionnement (notamment grâce aux TP de POOIG) mais malgré cela nous avons rencontré quelques problèmes qui ont assez freiné le développement de l'IG, car l'utilisation de cette bibliothèque demande une certaine rigueur au niveau du paramétrage des différents

éléments. En effet nous voulions proposer un affichage dynamique en fonction de la taille de la fenêtre du joueur, de façon à ce que tous les éléments de l'UI soit à bonne échelle. Cela a été très pénible à paramétrer car nous devions calculer, par exemple, la longueur et largeur de différents *JPanel* en fonction de la taille de l'écran, la taille des *JPanel* dans lesquels ils se trouvaient... Les *JLabels* nous ont également posé problèmes pour la mise à l'échelle car ils ne fonctionnent pas exactement comme les *JPanel*, mais nous avons finalement trouvé des alternatives.

2.5 Pistes d'extensions

- Ajout d'objets : potion de régénération des points de vie, rappel (réanimer un Pokéémon en lui redonnant la moitié de ses points de vie), antidotes pour guérir de la paralysie, du poison ou de brûlure.
- Interface permettant aux joueurs de composer leur équipe et de choisir le terrain.
- Ajout d'autres type de cases.
- Ajout d'autres Pokémons.
- Sauvegarder une partie en cours et pouvoir choisir quel sauvegarde utiliser pour jouer
- Ajout des capacités spéciales pour les Pokémons leur permettant d'avoir des effets passifs en plus.
- Ajout d'effets visuels et sonores pour les attaques.
- Complexification des calculs de dégâts des attaques par rapport à leur puissance , et des statistiques spéciales sur le Pokéémon en séparant par exemple les stats de point d'attaque en physique et magique, et en ajoutant des points de défense physique et magique.
- Complexification du système de tour et des coups critiques en choisissant l'ordre de chaque tour selon la stat' de vitesse du Pokémon et non selon le joueur.

3 Rôle(s) de chacun et difficultés rencontrées

3.1 GHANEM Wissam et TOKO Nawfel

Nous avons décidé de travailler sur la partie modèle du jeu. Pour ce faire on a commencé par créer les classes principales à savoir Pokemon, Joueur, Plateau, Case, Type, Attaque et Joueur M. On a tout d'abord attribué aux Pokémons tous les attributs qu'on puisse imaginer. On a ensuite donné au joueur une linkedlist de Pokémons et on a créé le plateau sous forme de tableau de tableau de case, les cases et Pokémons possédant eux-mêmes un type.

Pour ce qui est du placement des Pokémons sur le plateau ou voulait dans un premier temps donner à chaque cases un Pokémon mais on a finalement décidé d'utiliser les « hashmaps » avec les Pokémons comme clé pour éviter les conflits et faciliter les changements de cases.

Pour la partie déplacement on a eu un peu de mal à trouver comment procéder, après discussion avec notre professeur, il nous a proposé d'utiliser la fonction « bfs » afin d'obtenir les cases autour du Pokéémon voulu. Ainsi, en collaboration avec nos camarades travaillant sur la vue, on a pu créer la fonction *déplacement* permettant au Pokéémon en fonction de son type et suivant un range donné de se déplacer.

Pour ce qui est de l'attaque on a créé une classe abstraite Attaque avec comme classe filles les différents types d'attaques : c'est ici que l'on a spécifié les faiblesses et les forces de chacun des types et on y aussi implémenté les coups critiques à l'aide d'une variable *random*.

Pour améliorer le projet on a voulu y implémenter de la musique. Pour ce faire, après plusieurs essais infructueux à utiliser la librairie swing et après accord avec notre professeur, on a décidé d'utiliser la librairie *JMF*. Notre classe Audio.java nous permet alors de lancer un son au lancement du jeu ou bien lorsque l'on effectue une attaque.

Vers la fin du projet on a essayé de trouver un moyen de réaliser une sauvegarde du jeu. L'idée était de créer pour chaque variable du modèle une variable qui, lors de la sauvegarde, prendrait les valeurs des variables actuelles et au lancement d'une nouvelle partie les rétablirai. Cependant, le processus étant très long et les examens approchants, on a décidé de laisser cette idée de côté.

3.2 AMARA Mouloud

Construire le plateau de jeux :

On commence la partie par créer une Vue dans Lanceur. Dans le constructeur de Vue, on crée un Contrôleur, c'est ce dernier qui s'occupe de créer le Terrain et le Jeux dans la partie Modèle. J'ai choisi cela pour faciliter le recommencement de la partie en nous permettant de créer un nouveau Contrôleur et de le passer en paramètre à la nouvelle vue pour qu'elle s'initialise en fonction du terrain et des Pokémons créés. Au début, Vue est un *JFrame* qui contient un *JPanel* d'accueil avec un bouton "Commencer". Dès que l'on appuie sur ce bouton, deux *JFrame* sont créées, un pour le premier joueur et l'autre pour le deuxième, le panel d'accueil n'est plus visible. Dans le constructeur de EcranJeux, on obtient la dimension de l'écran et on dimensionne le *JFrame* du premier joueur dans la moitié gauche et celui du deuxième joueur dans la moitié droite de l'écran. Pour créer le terrain dans Vue, on met un *GridLayout* dans la moitié droite du *JFrame* avec les mêmes hauteurs et largeurs que le terrain dans la partie Modèle. Pour ajouter les Tiles au plateau de jeux qui sont des *JPanel* avec l'image de la case, la Vue demande au Contrôleur, pour chaque coordonnée, les chemins d'images de la case correspondante. Dans Case, le chemin de l'image pour sélectionner la case (on peut déplacer un Pokéémon dessus) et celui pour sélectionner la case en rouge (on peut attaquer le Pokéémon se trouvant dessus) sont choisis en fonction du type de la case qui est un *enum*. On aurait pu utiliser une classe pour chaque type de terrain qui hérite de terrain,

mais on utilisé l'enum, car c'est plus compacte, vu que les seules différences entre les terrains sont les images et les Pokémons qui peuvent s'y déplacer. Dans l'EcranJeux du deuxième joueur, j'ai commencé par insérer des Tile de coordonnées (0, 0) puis (0, 1) jusqu'à (height-1, width-1). Par contre dans l'EcranxJeux du premier joueur, j'ai commencé par insérer le Tile de coordonnées (height-1, width-1) puis (height-1, width-2) jusqu'à (0,0). J'ai fait cela afin que chaque joueur ait ses Pokémons affichés en bas de sa fenêtre.

Sélectionner les Pokémons qui peuvent être déplacé :

Quand un Pokémon est déplacé, il est ajouté dans un ensemble de Pokémons. Pour obtenir la liste des coordonnées des Pokémons qui peuvent être déplacés, on parcourt toute la liste des Pokémons du joueur, si un Pokémon ne se trouve pas dans l'ensemble des Pokémons déplacés, on ajoute ses coordonnées à la liste. Dans Vue, les Tiles correspondants à la liste des coordonnées sont colorés en bleu dans le *JFrame* du joueur actuel. A la fin du tour d'un joueur, l'ensemble des Pokémons déplacés est vidé, puis on passe à la liste des pokémons de l'autre joueur.

Récupérer les cases où le Pokémon peut se déplacer :

Sur ce lien, figure une implémentation de l'algorithme BFS qui parcourt toutes les cases d'un tableau à deux dimension à partir d'une case donnée. J'ai modifié sa fonction `isValid` pour qu'elle tienne compte du nombre maximum de cases que le pokémon peut parcourir et vérifie si le type du Pokémon lui permet d'aller sur la case. J'ai fait en sorte aussi que le Pokémon puisse sauter sur un Pokémon allié, mais non sur un Pokémon ennemi. Une liste de coordonnées est transmise à la Vue qui elle remplace les images des Tiles correspondantes par la même image colorée en bleu, uniquement dans la fenêtre du joueur à qui appartient le Pokémon.

Affichage des statistiques des Pokémons :

La moitié gauche de chacun des *JFrame* est divisé en deux, une partie pour l'affichage des stats, et l'autre pour les boutons. La partie en haut comporte un *JLabel* qui indique si c'est au joueur 1 ou au joueur 2 de jouer et deux *JScrollPane* mis côte à côte, l'un pour les stats des Pokémons du joueur 1 et l'autre pour ceux du joueur 2. Lors du tour d'un joueur et lorsqu'il met sa souris sur un Pokémon, le scroll bouge pour pouvoir voir les stats correspondantes aux Pokémons qui sont sélectionnés avec un cadre vert pour les Pokémons du joueur, rouge pour les Pokémons ennemis. Malheureusement, quand on passe la souris sur un panel des stats, le Pokémon correspondant n'est pas sélectionné, car je n'avais pas eu le temps pour le faire. Par contre, dans chaque *JFrame*, les Pokémons du joueur auquel appartient ce dernier sont sélectionnés avec un cadre vert, ce qui permet au joueur de différencier ses Pokémons de ceux du joueur adverse. Quand un Pokémon a un effet (peur, brûlure, paralysie...), un logo est affiché dans ses stats. Si le joueur passe son curseur sur le logo, un *JScrollPane* qui donne une description de l'effet s'affiche en haut, qui disparaît aussitôt que le joueur enlève sa souris. Quand la description est longue et que le joueur a besoin d'utiliser le scroll pour tout lire, il n'a qu'à appuyer sur le logo avec sa souris. Dans ce cas il faut rappuyer sur le logo pour faire disparaître la description.

Récupérer les cases qu'un Pokémon peut attaquer :

Dans Terrain, il y a une fonction qui attend les coordonnées de la case du Pokémon qui attaque et l'attaque choisie. Chaque attaque a une distance maximale d'attaque et peut, soit attaquer toutes les cases jusqu'à cette distance, soit attaquer la case qui se trouve exactement à cette dernière. Il y a des attaques qui peuvent passer sur des Pokémons ennemis ou alliés (par exemple : attaque éclair) et d'autres qui ne peuvent pas. Une liste de coordonnées est passée à la Vue qui remplace les images des Tiles correspondantes par la même image coloré en rouge dans les deux fenêtres. J'ai décidé de colorer les cases qui peuvent être attaquées même dans la fenêtre du joueur attaqué pour qu'il sache quels Pokémons sont en danger.

3.3 LE NINIVEN Adrien

Pour ma part je me suis surtout occupé de la partie Vue. Il fallait au début modéliser les Tiles du terrain en y insérant les différentes images correspondantes à différentes textures pour varier les types de terrain. J'ai eu quelques difficultés au début car n'étant pas à totalement à l'aise avec la bibliothèque Swing, j'oubliais de paramétriser certains attributs élémentaires du JPanel ce qui avait pour cause de ne provoquer aucun affichage, ce qui est très frustrant au début.

Mais grâce à l'aide de mes camarades j'ai appris de mes erreurs et me suis donc occupé de la base du panneau des statistiques des Pokémons. Pour cela je me suis beaucoup inspiré de ceux qu'on retrouve dans les différents jeux Pokémons. Le plus facile a été de récupérer les informations (points de vie, nom du Pokémon...) mais le plus dur a été de positionner chaque élément du JPanel de façon précise et à bonne échelle, ce qui demande beaucoup de tests. J'ai ensuite remplacer le nombre de points de vie par une barre de vie, qui varie en temps réel et change de couleur selon les points de vie du Pokémon.

Je me suis également occupé des différents terrains. Comme les *TypeCase* sont des enums, j'ai modifié le terrain que nous avions par défaut, qui est représenté par un tableau de *TypeCase* en faisant en sorte que chaque type ne soit pas trop désavantageux (par exemple un terrain contenant majoritairement de l'eau serait très désavantageux pour un Pokémon de type feu au niveau de ses déplacements).

Pour finir, je me suis occupé du contenu du panneau des logs. Ici, contrairement au panneau des statistiques des Pokémons, récupérer les variables était plus compliqué car notre projet était déjà assez dense au niveau du nombre de classes, il ne fallait pas "casser" l'architecture en important directement les informations du modèle dans la partie Vue. J'ai donc créé plusieurs *getters* afin de ne pas écrire à répétition le pointage des différentes variables.

Ma dernière tâche a été de m'occuper du diagramme des classes approfondi (réalisé sur InDesign) ainsi que de la présentation du rapport (réalisée en L^AT_EX).

3.4 TRAN Henri Bergson

Mon rôle principal a été d'aider sur toutes les parties du code si quelqu'un avait un problème, je travaillais sur la partie de Vue au début du projet puis je suis allé sur la partie du modèle pour compléter ce que Nawfel et Wissam avaient commencé pour la classe abstraite Pokémon, Attaque et la classe Audio.

J'ai ensuite continué à peaufiner et ajouter les différentes sous classes d'attaque, par exemples les Types d'attaque puis les attaques individuels qui ont des portées d'attaques et effets différents les unes des autres. Un exemple de cela est l'attaque Eclair qui permet d'attaquer à 1 ou 2 cases de distance, et qui a une chance (10%) de paralyser la cible lui réduisant sa distance de déplacement de 1 et fait en sorte qu'il ait 25% de chance de rater son attaque tant qu'il est paralysé. D'autres effets spéciaux ont aussi été implémentés avec des effets différents comme la brûlure , le gèle , la peur , la confusion ou le poison.

Les dégâts des attaques dépendent du type de l'attaque et celle du Pokémon attaqué et sont codés dans chaque sous classe Attaque[Type]. Chaque fois qu'un Pokémon attaque , on prend en compte si le Pokémon est confus, paralysé et si il réalise un coup critique car chaque cas peut changer radicalement les dégâts que provoque le Pokémon.

J'ai ensuite fait les boutons d'attaques qui sont nommés selon les attaques que chaque Pokémon possède et nous permet de voir la description et les cases que l'attaque peut toucher quand on passe notre curseur dessus. Un des problèmes rencontrés sur l'effet *hover* des boutons était lorsque l'on cliquait sur le bouton, il ne comptait pas comme si on sortait du bouton et donc ne cachait pas la description de l'attaque. Il a fallu ajouter un boolean qui vérifiait si on avait fait un click sur le bouton ou non et affiche la description tant qu'on avait pas cliqué donc quand le boolean était false.

L'ajout de l'audio avait également posé initialement quelques problèmes de fichiers mal initialisés car la bibliothèque n'accepte que des fichiers de type .wav, qui n'existent que pour les anciens effets sonores des jeux Pokémon originaux.

Enfin, j'ai décidé d'ajouter des *randomizers* pour les stats possibles de points de vie et d'attaque pour chaque Pokémon, leur rareté , et d'ajouter un système de Pokémon “shiny” ou “chromatique” en français indiquant si le Pokémon est d'une couleur différente de la normale. On a choisi de mettre le taux d'apparition de shiny a 5% pour ne pas trop impacter les parties car on a donné aux Pokémon shiny 30 points de vie et 5 points d'attaque en plus pour récompenser la chance d'avoir obtenu un Pokémon très rare.

J'ai aussi importé, modifié et redimensionné la plus part des images qui sont inclus dans le dossier ressources selon les besoins des autres membres du groupe et du projet. J'ai également aidé Adrien à faire en sorte que les attaques et effets affichent tous les détails dans le panneau des logs.

Les difficultés rencontrées étaient principalement sur les cases possible à attaquer , mais Mouloud nous a aidé pour cette fonction car il s'était occupé de la fonction de déplacement du Pokémon qui ressemble beaucoup à celle des choix de cases d'attaque. Mais aussi pour les différentes implémentations des effets sur les Pokémon affectés, car il fallait avoir un ordre précis des actions et des fonctions à appeler pour que le modèle et la vue fonctionnent sans avoir de différences entre eux.

4 Conclusion

En conclusion ce projet nous aura été bénéfique pour chacun. Il y avait une bonne ambiance durant En effet nous étions régulièrement en contact afin qu'un membre du groupe puisse en aider un autre lorsqu'il est bloqué sur une tâche par exemple. De ce fait même si certains membres possédait certaines lacunes, nous arrivions à nous répartir les tâches raisonnablement et à en apprendre chacun des autres.

5 Diagrammes des classes

Le premier diagramme est minimaliste afin de présenter facilement les différentes liaisons entre les classes pour différencier leur rôle. Le deuxième diagramme présente les attributs des différentes classes, les méthodes de certaines classes pertinentes ainsi que les différents héritages de classe.

