Première partie

Résumé du projet de langage C : Poketudiant

Nicolas Serf / Gabriel Révillion

Chapitre 1

Guide d'utilisation

1.1 Compilation et execution

Un simple **make à la racine du projet** permet de compiler l'ensemble du projet. Pour l'exécution, l'exécutable produit par le makefile s'appelle 'executable'. Ainsi pour lancer le programme, il suffit simplement de saisir la commande **'./executable'** à la racine du projet.

1.2 Interpreteur de commande

Les différentes commandes présentent dans le polycopier de projet sont utilisables dans le programme. Une sécurité est mise sur la saisie des commandes pour permettre de conserver un programme cohérent quand les saisies de l'utilisateur ne sont pas correctes. De plus, important, pour permettre de quitter proprement le programme sans fuite de mémoire, la commande **'exit'** a été ajoutée à la liste des commandes possibles.

Pour la partie 2 du projet, 4 nouvelles commandes ont été ajoutées qui sont : z, s, q, d, qui, respectivement, permettent d'avancer vers le nord, le sud, l'ouest et l'est.

1.3 Combats

L'interpréteur lors des combats est très simple, il faut simplement suivre pour les actions voulant être réalisées, le menu qui donne la bonne valeur à saisir.

Lorsqu'un de vos pokétudiants meurt, la programme va vous afficher les pokétudiants en vie restants, il faudra rentrer l'ID du pokétudiant que l'on souhaite faire rentrer en jeu. Si tous vos pokétudiants sont morts, alors le programme vous affiche que vous avez perdu la bataille et le programme se ferme.

Lors d'un changement de pokétudiant, l'interpreteur va réagir différemment selon certains critères. Quand votre pokétudiant meurt et que le programme vous demande de rentrer un nouveau pokétudiant, vous n'avez pas le choix et vous devez choisir un nouveau pokétudiant pour combattre (si il en reste, sinon la défaite est détectée). Toute autre valeur d'ID autre qu'un pokétudiant valable sera refusée. Lorsque vous choisissez de changer de pokétudiant volontairement alors que le votre est toujours en vie, alors l'interpréteur va accepter une nouvelle valeur qui est la valeur **'-1'**. Celle-ci permet d'annuler le changement de pokétudiant. Dans ce cas, votre tour n'est pas perdu et vous pouvez choisir une autre action.

Chapitre 2

Topo du premier rendu de projet

2.1 Les choix

Le projet s'est articulé autour de la modularité et de l'évolutivité du projet. Ainsi, avant même de commencer à coder les fonctionnalités propres à Pokétudiant, nous avons mis en place plusieurs structures génériques telles qu'une table de hachage gérant les collisions avec des listes chainées et un conteneur générique pouvant être dynamique ou statique et pouvant disposer ou non de la gestion de mémoire de ces élements. Pour faire un point objectif, ces structures ont permis par la suite d'être assez productif en disposant de structures de données génériques pour gérer ce que l'on souhaitait rapidement. Cependant il faut noter que nous aurions très bien pu nous en passer et simplement créer plusieurs structures de données répondant au même rôle mais chacune spécifique à un type de données.

La modularité et l'évolutivité du programme s'appuie essentiellement sur les defines utilisés qui permettent facilement de modifier le comportement du programme, comme par exemple le nombre maximum de pokétudiants dans une équipe ou encore le niveau maximum que ceux-ci peuvent avoir. Mais surtout, cela se trouve dans le chargement dynamique des entités du jeu. En effet, toutes les entités du jeu, c'est à dire les différents pokétudiants qui existent, les différentes attaques et les différentes faiblesses se trouvent dans des fichiers de configuration qui sont chargés à la création du programme. Ainsi il est aisé de créer de nouveaux pokétudiants, de nouvelles attaques et de nouveaux types. Ainsi si l'on souhaite créer un nouveau type, il suffit simplement d'ajouter dans le code une nouvelle valeur d'énumération représentant ce type et de l'ajouter dans les deux fonctions du code C permettant la transition enum¡-¿string. Pour ajouter des pokétudiants, des attaques ou des faiblesses, il suffit simplement de suivre la syntaxe utilisée dans les fichiers pour créer de nouvelles entités.

C'est ainsi qu'on peut comprendre l'utilité des tables de hachage. Celles-ci permettent de stocker les caractéristiques de base d'un pokétudiant ou d'une attaque. Ensuite le module chargé de la création des pokétudiants ou de son évolution va simplement aller chercher dans la table de hachage via le nom, les caractéristiques de base d'un pokétudiant pour pouvoir le créer.

Précisons un dernier point dans les choix, concernant la méthode de travail en groupe que nous avons adopté. Pour une facilité de communication et transfert de fichier, nous avons utilisé git via la plateforme github permettant de travailler chacun de son côté et

de pouvoir joindre nos différents fichiers sans soucis.

2.2 Les difficultés

Cette première partie du projet ne nous a pas réellement causé de grosses difficultés. Nous avons grâce à Github pu travailler indépendemment sur nos fichiers et ensuite fusionner nos travaux très facilement. La principale difficulé se trouve dans la réflexion que nous avons fait sur l'architecture du programme. Les différents modules présents, l'agencement du projet et le chargement des données du jeu permettant d'être aussi évolutif que possible. Nous avons aussi eut quelques soucis avec le buffer qui nous obligait à rentrer des espaces ou des retours chariot parfois incompréhensible à la suite d'une commande venant d'être executée. Cependant ces problèmes ont normalement été réglés.

2.3 Fonctionnels

A l'heure actuelle et au vu des différents tests que nous avions réalisés, le programme posséde toutes les fonctionnalités attendues qui fonctionnent selon le comportement demandé. Le programme, s'il détecte une incohérence dans les commandes de l'utilisateur se chargera de le prévenir que celles-ci ne sont pas bonnes.

2.4 Non fonctionnels

Rien à signaler

Chapitre 3

Topo du deuxième rendu de projet

3.1 Les choix

La deuxième partie du projet étant moins dense niveau contenu, il n'y a pas vraiment eu de choix à réaliser pour cette partie du projet. La carte est réalisée de manière classique via un chargement de fichier précisant dans les 2 premières lignes la taille de la carte et ensuite la carte est représentée sous forme de chaine de caractère avec comme expliqué plus en détail dans la partie difficulté, le fonctionnement du parsing. Le chargement des dresseurs constituant le conseil des diplômes est fait via un fichier se trouvant à part du fichier de la carte.

3.2 Les difficultés

La partie nous ayant causée le plus de difficultés est la mise en place de zone de niveau pour les régions sauvages et la représentation d'ennemis. Nous avions pensé dans un premier temps, charger indépendamment toutes les régions sauvages et ensuite procéder à un parcours des cases voisines d'une case choisie au hasard pour affecter un niveau aléatoire mais cela provoquait des maps changeant à chaque execution est un temps de calcul beaucoup plus grand. Nous avons ainsi opté pour une représentation du niveau de la région sauvage directement dans le fichier en spécifiant le niveau. Pour la représentation des ennemis qui représentent le conseil des diplômes, nous avons transformé notre parsing pour que celui-ci soit fait avec des chaines de caractères, ainsi dans le fichier texte de la carte. Nous avons des chaines de caractère avec directement le nom du dresseur du conseil des diplomes, on est ainsi certain que celui-ci ne change pas de place et il est facile de le placer et d'intéragir avec lui lorsque l'on rentre dans sa zone.

3.3 Fonctionnels

Nous nous sommes rendus compte que nous avions oublié une fuite mémoire dans la première partie lorsqu'un pokémon évolue, le programme réalise un affichage pour donner l'ancien nom du pokétudiant vers le nouveau nom du pokétudiant. Seulement nous avions oublié de free l'ancien nom du pokétudiant. Cette erreur étant maintenant corrigée, le programme posséde normalement toutes les fonctionnalités attendues partie 1 et partie 2 comprises et semble fonctionner sans crash ou de fuite de mémoire.

3.4 Non fonctionnels

Rien à signaler

3.5 Répartition du travail

La répartition du travail sur l'ensemble du projet a été réalisé équitablement avec 50/50 pour chaque binôme.