Rapport du projet de C++ Pokemon, la chasse aux spectres !

réalisé par
Laurent Dang-Vu
Najwa Moursli

Polytech Sorbonne

Spécialité

Mathématiques Appliquées et Informatique Numérique

3ème année

Paris, France 2020

Table des matières

1 Description de l'application développée

Le projet consiste à développer un jeu Pokemon quelque peu atypique. En se restreignant au thème de la fumée, nous nous sommes inspirés de la fameuse tour Lavanville de la première génération de pokémon. Cette tour est connue pour abriter de nombreux pokémons de type spectre qui demeurent invisibles au milieu de la fumée... Mis à part ce fait connu, quelles autres suprises surprenantes nous réserve t-elle ?

Notre jeu proposera donc au joueur de s'aventurer dans la tour Lavanville. Il devra se frayer un chemin jusqu'au pokemon terrifiant qui est devenu maître incontesté des lieux.

Par conséquent, comme dans tout jeu Pokémon, le joueur devra se déplacer à travers une carte pour affronter le pokémon présent sur ma carte. L'application s'arrêtera en cas de victoire contre le boss ou lorsque tous ses pokémons seront hors-jeu (perte du combat). Voivi un exemple du lieu où se déroule le combar, un combat commence lorsqu'un pokemon se déplace sur les cases adjacente à sa position (cases en marrons sur l'image) :

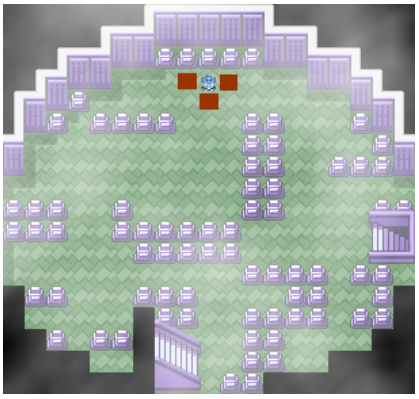

FIGURE 1 – Map de combat

Le joueur aura une équipe de pokémons prédéfinie tous au niveau 50 et qui ne pourront pas passer à un niveau supérieur comme ses adversaires.

Afin de réaliser se projet nous avons décider de simplifier le jeu Pokémon par rapport à l'original.Voici les simplificatiosn qui ont été effectuées :

- Le dresseur ne possède qu'un pokemon car on veut eviter l'option d'échange de pokemon
- Création d'une map avec seulement du carrelage, pas d'obstacles ni de murs car on veut eviter de devoir coder des couches
- Toutes les categories d'attaque ne sont pas présentes comme les attaques qui changent le statut de pokemon (paralysie, sommeil, brulure, gel)
- Ne pas afficher des animations ni des bulles de dialogue interactives (texte qui defile)
- Se limiter à 3 pokemon ennemis à affronter

Afin de modéliser ce jeu nous avons choisis d'utiliser SFML qui est une interface de programmation destinée à construire des jeux vidéo ou des programmes interactifs. Elle est écrite en C++.

2 Procédure d'installation et d'exécution du code

Pour exécuter le code, il faut installer la librarie SFML-2.5.1.La version de SFML que vous souhaitez installer est disponible dans les dépôts officiels, alors installez-la simplement avec votre gestionnaire de paquets. Par exemple, sous Debian vous feriez : **sudo apt-get install libsfml2.1-dev** Si vous avez installé SFML dans un chemin non-standard, vous devez indiquer au compilateur où trouver les en-têtes SFML (les fichiers .hpp) : Dans le Makefile ceci correspond à ligne : **INCDIR=-I//home/sasl/shared/main/c++/SFML-2.5.1/include** Ici, /home/sasl/shared/main/c++/SFML-2.5.1 est le répertoire dans lequel est copié SFML. Puis, vous devez lier le fichier compilé aux bibliothèques SFML afin de produire l'exécutable final. SFML est composée de 5 modules (système, fenêtrage, graphique, réseau et audio), et il y a une bibliothèque pour chacun. Pour lier une bibliothèque SFML, vous devez ajouter "-lsfml-xxx" à votre ligne de commande, par exemple "-lsfml-graphics" pour le module graphique (par rapport au nom du fichier correspondant, le préfixe "lib" et l'extension ".so" doivent être omis). Dans le Makefile ceci correspond à la ligne **CLIBS= -L/home/sasl/shared/main/c++/SFML-2.5.1/lib -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio**
Après s'être placé dans le répertoire où se trouve le projet il rest plus qu'a taper la commande **make -f Makefile** pour compiler touts les fichiers et . **Pokemon** pour exécuter l'executable qui vient d'être créer.

3 Utilisation des contraintes

Notre application se composent en trois fichiers headers *global.hh, move.hh et pokemon.hh* ainsi quatres fichiers sources *global.cc, move.cc, pokemon.cc et deux fichiers main.cc.*

3.1 Niveaux de hiérarchie

L'application comporte plusiers niveaux de hiérachie dû à de nombreux hériatges entres les différentes classes. Tout d'abord dans le fichier *pokemon.hh*, on a définit une classe mère Pokemon pour ensuite à partir de ses attributs, constructeur et méthodes définir tous les différents Type de Pokemon existants *Fire, Fight, Ghost, Fairy, Psychic, Grass,Water, Electric, Steel, Poison, Normal, Ice, Ground, Flying, Dragon, Dark et Bug*, comme on peut le voir sur le schéma :

FIGURE 2 – Representation des différents Type de Pokemon avec leur faiblesses à gauche, les types concernés au milieu et quel type peuvent ils attaquer à droite

Comme vous pouvez le voir dans la classe Pokemon du fichier *pokemon.hh* un Pokemon est constitué d'un nom un type un niveau, de points de vie, d'attaque/défense, d'une vitesse et d'un ensemble d'actions possibles stocker dans un vecteur de Type Move, avec les fonctions getters/setters ont récupère toutes ses informations et il ne reste plus qu'a initialisé le constructeur des classes filles hérité de la classe Pokemon avec leur propre valeurs pour chaque attribut.

Un autre héritage à quatre niveau de hiérarchie est présent dans le jeu, cet héritage est complexe car il fait appel à la notion d'héritage multiple et virtuel. Cela concerne les différentes actions d'un Pokemon et leur impact sur celui-ci et son adversaire. Voici un schéma illustrant les liens d'éhritage entre les différentes classes :

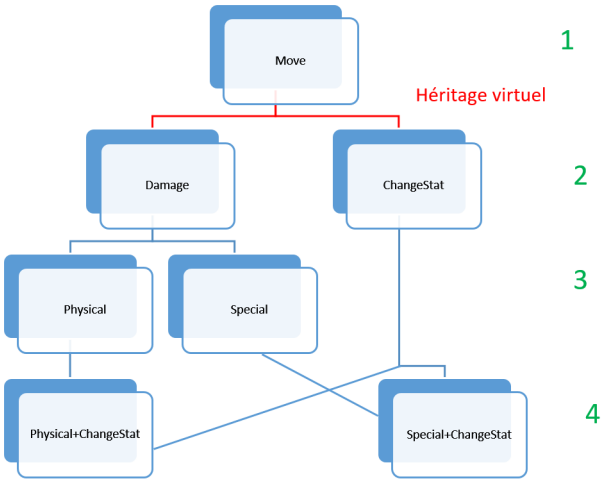

FIGURE 3 – Système héritage multiple et virtuel

Dans le fichier *move.hh*, après avoir procédé à une déclaration anticipée de la classe Pokemon pour indiqué que cette classe est connu et qu'il y a une référence croisé avec la classe Move, on observe l'héritage des différentes classes de ce fichier.

1. Move : les différentes actions d'un type de Pokemon par exemple :
   — **std : :string m_name** : *"lance flamme"*
   — **Type m_type** : *Fire*
   — **std : :vector<MoveCategory> m_categories** : *Special, StatEnemy*
2. Damage : les attaques qui infligent des degats. On déclare cette classe directement après la classe Move puis *Class Damage : **virtual public** Move*. Ici, le constructeur de Damage appelle celui de Move afin d'initialiser correctement les attributs de Move présents dans Damage. La classe Move doit arriver en première position dans la liste des super classes et l'héritage sur Move doit de nouveau être virtuel. Cette construction garantit que les attributs de Move ne seront présents qu'en un seul exemplaire, et ce, directement depuis Move et le constructeur de Move est explicitement appelé dans celui de Special+ChangeStat et Physical+ChangeStat assurant ainsi l'initialisation correcte des attributs hérités de Move. Conséquemment, les appels au constructeur de Move depuis ceux de Damage et ChangeStat ne seront pas exécutés.

3.2 Surcharge d'opérateurs

Nos principales surcharges d'opérateur sont celle d'affichage du flux
std : :ostream &operator<<(std : :ostream &flux,Type const& t). Cette fonction permet d'afficher à l'utilisateur sur l'invite de commande les diférents choix de Type, Actions et de surveillers les points de vie des pokemon durant le jeu et celle en entrée inclut dans <iostream>.

3.3 Conteneurs STL

Dans l'ensemble du programme nous avons utilisé de nombreux conteneurs STL pour simplifier la manipulation des variables tels que *vector,map,string* :

- Dans la classe Pokemon, **std : :vector<const Move*> m_moves** est un vecteur de pointeurs constants de type Move stockant les différentes actions possible d'un pokemon. Grâce à la méthode **std : :string Pokemon : :movesetToString()std : :string s = "Moveset :"+m_name ;**
  s += " 1. " + m_moves[0]− >get_name() ... on récupère les actions stockés dans ce vecteur an passant par une méthode get.
- **extern const std : :map<Type,std : :vector<Type>> weaknessesTable** la clé Type est utilisée pour trier et indentifier chaque élements de la liste enum pendant que vector<Type>stock le contenu associé à cette clé. Tout ceci enmagasiné dans la valeur weaknessestable (se rèfère au schéma de la FIGURE 2). Utilisé dans la fonction **apply_move**, par la commande **std : :vector<Type> weaknesses = weaknessesTable.at(defenderType)** on vérifie si celui qui reçoit l'attaque par un pokemon fait partie des faiblesses de ce type pour ensuite attribuer un dommage proportionnel.

3.4 Fonction virtuelle

1. **virtual void apply_move(Pokemon& attacker,Pokemon& defender)** est une fonction qui permet d'émettre une action (attaque/deplacement/defense) envers un autre pokemon. Cette fonction est virtuelle car elle prend en argument des références attacker et defender de type Pokemon qui détermineront le type d'action en temps réel.
2. **virtual void draw(sf : :RenderTarget& target, sf : :RenderStates states) const** est une fonction inhérente à SFML qui permet de dessiner tous les objets de types SFML à l'écran tel que *window,texture ....* Comme on a créé une nouvelle classe Tilemap qui dessine la map avec des cases, on a défini draw pour cette classe précise.

4 Les parties de l'implémentation dont vous êtes les plus fières

4.1 TileMap

Afin de créer un jeu en 3D et de pouvoir introduire dans un premier temps des animations simple tel que le déplacement d'un personnage sur une plateforme 2D, nous avons utilisé le pricipe de TileMap à l'aide de Tilesets. On crée une classe qui encapsule une tilemap, c'est à dire un niveau composé de tuiles. Le niveau complet sera contenu dans un unique tableau de vertex (ensemble de point graphique ici on utilise des quad : 4 vertex), ainsi il

sera extrêmement rapide à dessiner. Il faut noter que cette stratégie n'est applicable que si tout le tileset (la texture qui contient les tuiles) peut tenir dans une unique texture. Sinon, il faudra au minimum utiliser un tableau de vertex par texture.

— on charge la texture du tileset
— on redimensionne le tableau de vertex pour qu'il puisse contenir tout le niveau
— on remplit le tableau de vertex, avec un quad par tuile
— on récupère le numéro de tuile courant
— on en déduit sa position dans la texture du tileset
— on récupère un pointeur vers le quad à définir dans le tableau de vertex
— on définit ses quatre coins(avec les indices du tableau quad)
— on définit ses quatre coordonnées de texture
— on applique la transformation
— on applique la texture du tileset
— on dessine enfin le tableau de vertex
— on définit le niveau à l'aide de numéro de tuiles
const int level[] = 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ... ;
— on crée la tilemap avec le niveau précédemment défini
TileMap map ; if (!map.load("tileset.png", sf : :Vector2u(32, 32), level, 16, 8)) return -1 ;

FIGURE 4 – Sprite personnage et Tileset de la Map

4.2 Combat avec gestion des évenements

Un combat ou dééplcement d'un joueur est assimilable à un enchainement d'évenement en SFML dont la syntaxe est bien précise : on crée l'évenement, on le récupère (appuyer sur clavier, souris ..) et on l'affiche :

1. **sf : :Clock clock ;** ***gestion du temps***
   sf : :Event event ; ***crétaion de l'évenement***
   while (renderWindow.isOpen())
   while (renderWindow.pollEvent(event))
   if (event.type == sf : :Event : :EventType : :Closed)
   renderWindow.close() ;
   Peter2.updateindex(map) ;
2. ***evenements lies au deplacement du joueur*** **bool collision = false ;**
   if (event.type == sf : :Event : :KeyPressed)*evenement = appuyer sur une touche*
   level[Peter2.getindX() +Peter2.getindY()*map.getxTiles()]*Le personnage se déplace*

3. **if (event.key.code == sf : :Keyboard : :S)** ***le point (0,0) est en haut a gauche***
   collision = Peter2.collision(DOWN, map, level, level2) ;
   Peter2.movedown(collision, clock) ;
   déplacement du personnage vers le bas quand il y a collision avec un obstacle sur la map

FIGURE 5 – Exemple de scène de combat

5 Diagramme UML

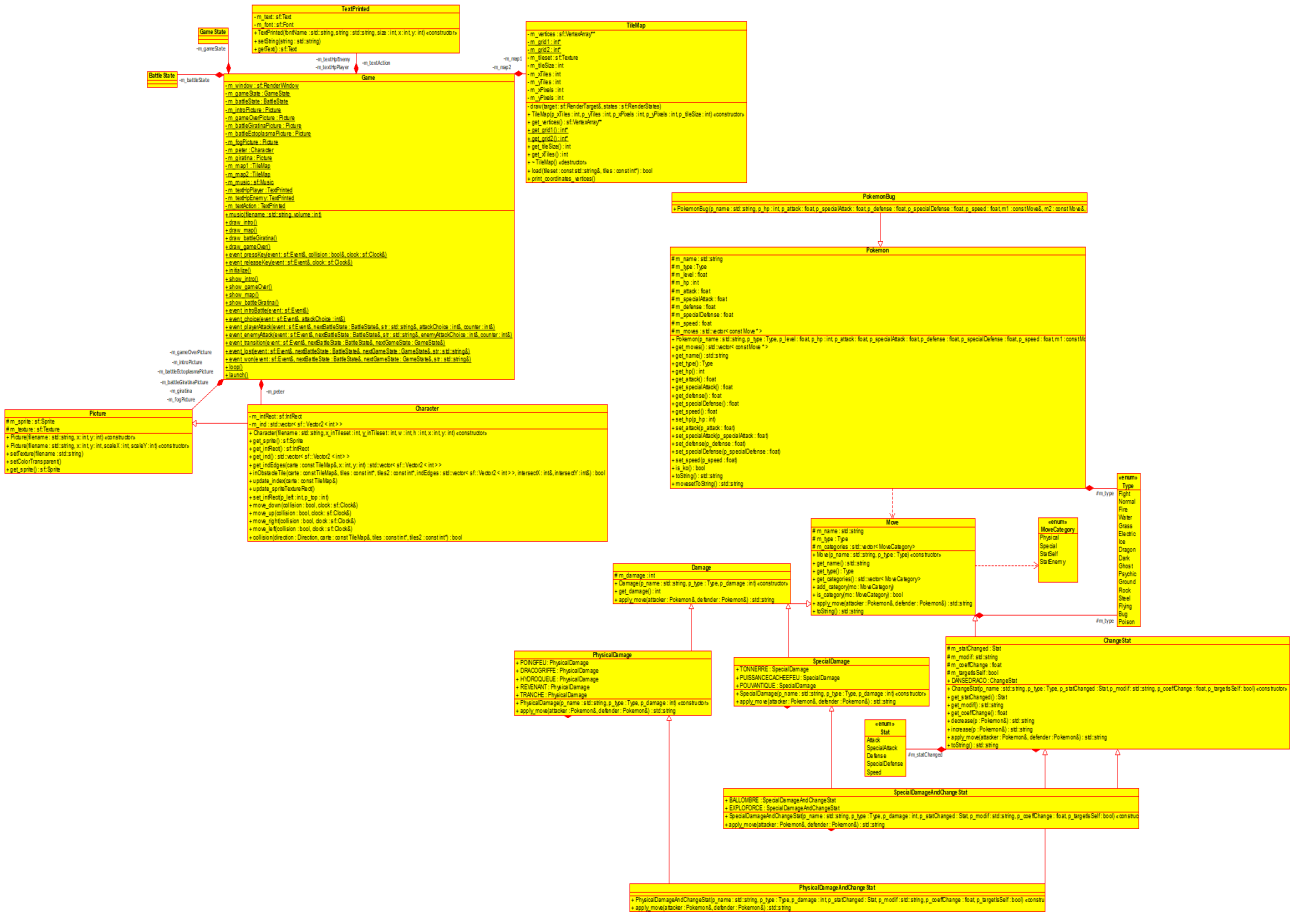

FIGURE 6 – Diagramme UML complet de l'application développée

6 Conclusion

Ce projet nous a permis de découvrir le monde du développement d'interface graphique tout en utilisant le langage C++. Nous avons appris aussi bien à mieux code en C++ qu'à soigner le graphisme du jeu tout en s'amusant et en créeant un application à laquelle nous aimons jouer et qui, nous l'espérons saura vous divertir.
Il persiste cependant à la fermeture du jeu un seg fault dont on ne trouve pas la cause. Nous vous invitons à regarder plus en détails notre code.

Références

1. **https ://www.gamefromscratch.com/post/2015/10/26/SFML-CPP-Tutorial-Spritesheets-and-Animation.aspx** (pour les manipulation de sprites)
2. **https ://www.sfml-dev.org/tutorials/2.5/graphics-vertex-array-fr.php**
3. **https ://openclassrooms.com/fr/courses/1515531-creez-des-applications-2d-avec-sfml/1516515-ajouter-de-la-musique-avec-le-module-audio**
4. **https ://www.pokepedia.fr/Portail :Accueil** (ressources pour nos Pokemon)