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

2ème 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 (proposé par le corps enseignant), nous nous sommes inspirés de la fameuse tour Lavanville de la première génération de pokémon. Cette tour est connue pour abriter de nombreux pokemon 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 le maître incontesté des lieux.

Par conséquent, comme dans tout jeu Pokémon, le joueur devra se déplacer à travers une carte pour affronter le pokémon légendaire sur ma carte. L'application s'arrêtera en cas de victoire contre celui-ci ou lorsque tous ses pokémons seront hors-jeu (perte du combat). Un combat commence lorsque le joueur se déplace sur les cases adjacente à la position du légendaire (cases en marrons sur l'image) :

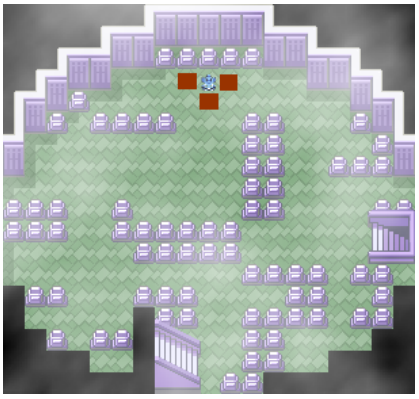

FIGURE 1 – Map de combat

Le joueur aura une équipe de pokémons prédéfinie tous au niveau 100 [1] qui ne pourront pas passer à un niveau supérieur comme ses adversaires.

Afin de réaliser ce projet, nous avons décidé de simplifier le jeu Pokémon par rapport à l'original. Voici les simplifications (non-exhaustives) effectuées :

- Le dresseur ne possède qu'un seul pokemon car on ne veut pas qu'il puisse changer de pokemon au cours du combat.
- Toutes les catégories d'attaque ne sont pas présentes, par exemple les attaques qui changent le statut d'un pokemon (paralysie, sommeil, brulure, gel, confusion) et celles qui font varier le taux de coups critiques [2].
- Nous ne prenons pas en compte les coups critiques qui nous obligeraient à faire intervenir des probabilités.
- Nous n'afficherons pas d'animations ni de bulles de dialogue interactives [3] (texte qui defile).

1. Niveau connu comme étant le niveau maximal et auquel les compétitions se jouent.
2. Le probabilité qu'a un pokemon d'infliger plus de dégâts que d'habitude avec ses attaques offensives
3. Comme lorsque le joueur dialogue avec des personnages ou fait face à un événement particulier

— Nous ne mettrons qu'un seul combat (pokemon à affronter : un pokemon légendaire mais pas de rencontres aléatoire habituelles [4]).

Afin de modéliser ce jeu nous avons choisi d'utiliser SFML, une librairie C++ connue destinée à construire des jeux-vidéo ou des programmes interactifs incluant des images.

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, installez-la donc 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 devrez 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 Description du programme

Notre application se compose de neuf fichiers headers *global.hh, global2.hh, move.hh, pokemon.hh, character.hh, picture.hh, text.hh, tilemap.hh, game.hh*, de huit fichiers sources *global.cc, global2.cc, move.cc, pokemon.cc, character.cc, tilemap.cc, game.cc, main.cc.*

—

3.1 Niveaux de hiérarchie

L'application comporte plusieurs niveaux de hiérachie dus à de nombreux héritages entres les différentes classes. Tout d'abord dans le fichier *pokemon.hh*, on a défini une classe mère Pokemon pour ensuite définir à partir de ses attributs, de son constructeur et de ses

4. Habituellement, des combats avec des pokemon sauvages se déclenchent aléatoirement au bout d'un moment lorsque le joueur se déplace dans certaines zones habitées par des pokemon (les hautes herbes, les forêts ou les bois, les grottes etc.

méthodes tous les différents types 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 types de pokemon avec leur faiblesses à gauche, les types concernés au milieu et envers quel(s) type(s) ils sont très efficaces [5]

Comme vous pouvez le voir dans la classe Pokemon du fichier *pokemon.hh*, un pokemon est constitué d'un nom, d'un ou plusieurs types, d'un niveau, d'un nombre de points de vie, de statistiques (attaque, défense, vitesse, etc.) et d'un ensemble d'actions possibles stockées dans un vecteur (le conteneur) de Type Move. Avec les accesseurs, on récupère toutes ses informations et il ne reste plus qu'à initialiser les attributs des classes filles héritées de la classe Pokemon avec les propres valeurs données en entrées à l'aide des mutateurs.

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

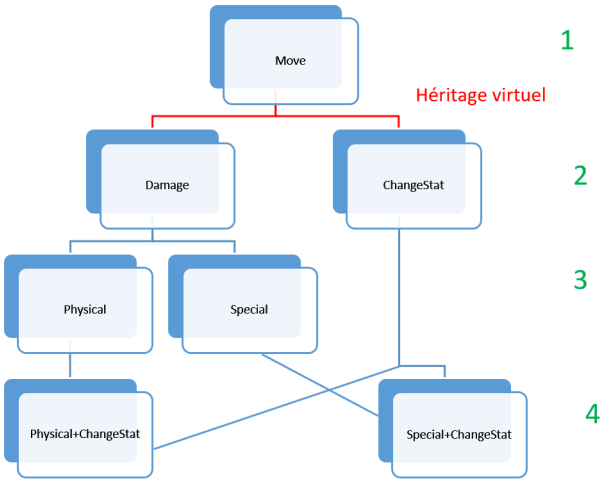

FIGURE 3 – Système héritage multiple et virtuel

Dans le fichier *move.hh*, après avoir procédé à une déclaration anticipée de la classe Po-

kemon pour indiquer que cette classe est connue et qu'il y a une référence croisée avec la classe Move, on observe l'héritage des différentes classes de ce fichier.

1. Move : c'est une action (attaque dans un sens général, correspondant plus à move en anglais, attaque étant un mot ambigu), par exemple on a pour l'attaque Lance-Flammes (attaque spéciale[6] offensive de type feu) :
   — **std : :string m_name** : *"lance flamme"*
   — **Type m_type** : *Fire*
   — **std : :vector<MoveCategory> m_categories** : *Special*
2. Damage : classe pour les attaques qui infligent des degats, les attaques offensives. 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. 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 surcharges d'opérateur sont celles d'affichage de 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.

std : :ostream operator¡¡(std : :ostream flux,MoveCategory const cat). Cette fonction permet d'afficher à l'utilisateur sur l'invite de commande les diférentess valeurs possibles de MoveCategory..

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 aux commandes **std : :string Pokemon : :movesetToString()std : :string s = "Moveset :"+m_name ;**
s += " 1. " + m_moves[0]− >get_name() ..., on récupère les attaques stockées dans ce vecteur an passant par un accesseur.

— Avec **extern const Avecstd : :map<Type,std : :vector<Type>> weaknessesTable**, la clé Type est utilisée pour trier et identifier chaque élement de la liste enum pendant que vector<Type> stocke le contenu associé à cette clé. Tout ceci est enmagasiné dans la valeur weaknessestable (se référer 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

6. Par opposition à attaque tout court qui signifie implicitement attaque physique

l'attaque d'un pokemon possède un type qui fait partie des faiblesses du type de l'attaque pour ensuite attribuer un bonus de dommage.

3.4 Fonction virtuelle

1. **virtual void apply_move(Pokemon& attacker,Pokemon& defender)** est une fonction qui permet d'émettre une action envers un autre pokemon. Cette fonction est virtuelle car elle prend en argument des références attacker (le pokemon qui attaque) et defender (le pokemon qui subit l'attaque) de type Pokemon qui détermineront l'effet de l'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, etc..* Comme on a créée une nouvelle classe Tilemap qui dessine la carte (la "map") avec des cases (les "tiles"), on a défini draw pour cette classe précise.

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

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 principe du 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.

Voici les différentes étapes à suivre pour créer le décor :

1. On charge la texture du tileset
2. On redimensionne le tableau de vertex pour qu'il puisse contenir tout le niveau
3. On remplit le tableau de vertex, avec un quad par tuile
4. On récupère le numéro de tuile courant
5. On en déduit sa position dans la texture du tileset
6. On récupère un pointeur vers le quad à définir dans le tableau de vertex
7. On définit ses quatre coins(avec les indices du tableau quad)
8. On définit ses quatre coordonnées de texture
9. On applique la transformation
10. On applique la texture du tileset
11. On dessine enfin le tableau de vertex
12. 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 ... ;

13. 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 déplacement d'un joueur est assimilable à un enchainement d'évenements en SFML dont la syntaxe est bien précise : on crée l'évenement, on le récupère (appuyer sur une touche précise du clavier, la 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

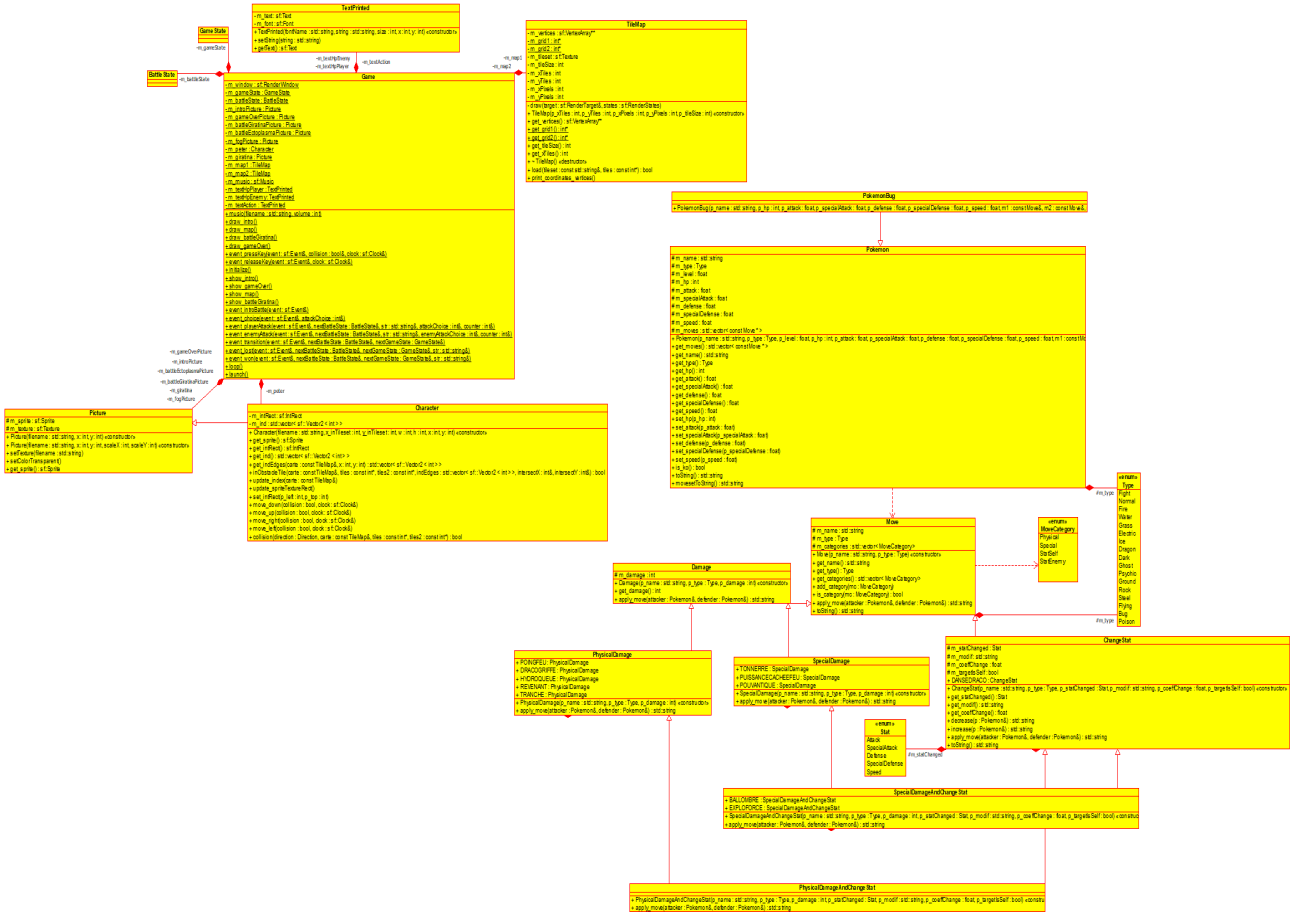

FIGURE 6 – Diagramme UML complet de l'application développée

6 Conclusion

Ce projet nous a permis de découvrir le langage C++, le monde du développement de l'interface graphique et la programmation orientée objet de manière ludique.

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)