Rapport C++

Alexis Fauxbâton, Rafael Carvallo, Emile Pichard

January 2022

1 Introduction

Dans le cadre du projet de fin de module sur le thème "There is no planet B", nous avons décidé de créer un jeu codé en C++ et utilisant la bibliothèque graphique consistant à faire se déplacer notre personnage principal (appelé "Maître") de salle en salle et à vaincre les ennemis présents en chacune d'elle grâce à son équipe d'alliés jusqu'à arriver à la dernière salle du jeu. Il pourra libérer certains alliés d'ennemis les retenant en otage. Le jeu est inspiré de jeux de combat au tour par tour se déroulant dans plusieurs petites cartes comme les jeux Pokémon.

2 Classes

Durant le développement de ce jeu nous avons dû implémenter 7 classes mères différentes ainsi que 6 classes filles au total. Cette partie a pour but de présenter une brève explication de la fonction de ces différentes classes.

2.1 Personnage

La classe "Personnage" a été implémentée dans le but de comprendre toutes les caractéristiques générales qu'un personnage (jouable ou non) peut posséder. Elle comprend entre autres des méthodes virtuelles lui permettant d'attaquer ou non un autre personnage. Cette classe mère abstraite est ensuite divisée en 3 classes filles non abstraites se nommant "Allie", "Ennemi" et "Maitre". Nous voulions rendre impossible une attaque entre personnages d'une même sous-classe empêcher la classe Maitre (personnage principal contrôlé par le joueur) d'attaquer d'autres sous-classes de personnages. C'est pourquoi nous avons choisi de rendre les méthodes d'attaque virtuelles. Grâce à la classe Personnage on peut avoir accès au nom du personnage, sa statistique d'attaque, de défense, de points de vie, son équipement ainsi que son sprite pour le dessiner à l'écran.

La classe fille "Allie" permet d'utiliser les mêmes actions que la classe mère Personnage mais permet de la différencier des autres classes filles "Ennemi" et "Maitre" sur le même niveau de hiérarchisation.

La classe fille "Ennemi" permet de déterminer et de savoir si un ennemi possède un prisonnier allié à recruter par le joueur.

La classe fille "Maitre" permet d'obtenir la liste d'alliés du joueur, accéder à leurs informations et permet d'y ajouter un allié supplémentaire.

2.2 Objet

La classe "Objet" a été implémentée afin de contenir toutes les caractéristiques propres aux objets pouvant être transportés par un personnage. Elle permet entre autres de savoir à quel point un objet améliore la statistique d'attaque, de défense et de points de vie du personnage la possédant, ainsi que de modifier ces paramètres On peut également obtenir le sprite de l'objet afin de l'afficher à l'écran. La dernière méthode implémentée est la surcharge de l'opérateur "=", permettant de copier plus facilement des objets (et de gérer la copie des attributs). Nous avons également implémenté 3 classes filles, se nommant "Arme", "Armure" et "Mystique". Ces classes filles sont identiques à la classe mère objet, mais permettent de faire une distinction plus facile entre les différents types d'objet que l'on peut ajouter dans l'équipement d'un personnage que l'on verra dans la partie suivante.

2.3 Equipement

La classe "Equipement" a été implémentée afin de permettre de gérer l'équipement de chaque personnage. Chaque personnage possède 1 emplacement d'objet de chaque type : "Arme", "Armure" et "Mystique". Cette classe comprend des méthodes permettant d'obtenir les informations sur l'arme, l'armure, et l'objet mystique de chaque personnage. Elle contient également des méthodes destinées à assigner un nouvel objet de chaque type dans chaque emplacement de l'équipement.

2.4 Map

La classe "Map" permet de gérer les caractéristiques propres à chaque carte dans laquelle le joueur peut se déplacer. Elle contient un vecteur d'ennemis représentant tous les ennemis présents sur la carte, le sprite à afficher à l'écran lorsque le joueur n'est pas en combat, le sprite à afficher à l'écran lorsque le joueur est en combat ainsi que les musiques correspondantes à chacune de ces situations. Elle contient également des pointeurs sur d'autres Map représentant les différentes cartes accessibles à partir de la carte actuelle, vers lesquelles le joueur peut se déplacer en empruntant des "portes", représentées par la classe "Porte" dont on parlera dans la partie suivante. Les méthodes contenues par la classe "Map" sont les différents "getters" et "setters" correspondant aux attributs cités précédemment, ainsi que des méthodes permettant d'ajouter des

ennemis ou d'en retirer de la liste d'ennemis de la zone (servant à l'initialisation de la carte et à la défaite d'un ennemi respectivement).

2.5 Porte

La classe "Porte" est une classe utile à la classe "Map" afin d'identifier des zones à l'intérieur desquelles le joueur pourra changer de carte. Pour ce faire, cette classe comprend en attributs les coordonnées x et y de deux points qui définissent une ligne qui, au passage du joueur, va le faire se téléporter dans la carte correspondante dans la boucle de jeu principale (définie dans la classe "Game" dont on parlera par la suite). Les différentes méthodes correspondant à cette classe sont les "getters" et "setters" des coordonnées des deux points définissant la porte ainsi qu'une méthode permettant d'afficher ces coordonnées.

2.6 Obstacle

La classe "Obstacle" est également une classe utile à la classe "Map" afin d'identifier cette fois-ci les zones inaccessibles par le joueur sur une carte. En effet, le joueur n'est pas par exemple censé pouvoir se déplacer dans des zones où il y a des arbres par exemple par soucis de cohérence de l'affichage. On ne voudrait pas par exemple voir le personnage principal affiché au-dessus d'arbres ou de structures du décor. C'est pourquoi l'on a besoin de définir des limites où il ne pourra pas accéder. Cette classe est composée de 4 attributs : Les coordonnées en X et Y du point de référence de l'obstacle, la longueur de l'obstacle et sa largeur. En effet, par soucis de simplicité, nous avons choisi de n'implémenter que des obstacles linéaires (forme rectangulaire), pour lesquels ces 4 paramètres définissent entièrement la zone d'effet de l'obstacle. Cette classe possède comme méthodes les getters et setters de ces 4 attributs, ainsi qu'une méthode destinée à détecter la collision de l'obstacle avec un allié passé en argument.

2.7 Game

La classe "Game" est la classe gérant le déroulement du jeu. Elle possède une seule méthode "run" prenant en arguments un vecteur de "Map*" représentant l'ensemble des cartes du jeu et donc les cartes auxquelles elles sont reliées. C'est dans cette méthode que la boucle de jeu est implémentée. Elle gère tous les évènements se déroulant dans le jeu avec par exemple le changement de carte, de musique, l'activation des touches gérant le déplacement du personnage, le passage au stade "en combat" ainsi que l'affichage de tous les éléments du jeu à l'écran. La boucle de combat est quant à elle implémentée et gérée dans la classe "Combat", dont on va parler dans la partie suivante, et qui est la classe chargée de gérer le déroulement des combats.

2.8 Combat

La classe "Combat" a été implémentée afin de gérer le déroulement d'un affrontement entre le joueur, ses alliés et un ennemis qu'il a rencontré dans une Map. Cette classe contient donc 3 arguments : un joueur qui est de la classe "Maître", un ennemi de la classe "Ennemi" et une map de la classe "Map".

Méthode "commencer_combat" : elle permet de lancer le combat entre le joueur et l'ennemi. Elle commence par initialiser toutes les choses nécessaires au bon déroulement du combat :

-On créée une window de la taille de l'image correspondant au sprite combat de la map chargée

-On positionnes les alliés au bon emplacement sur la map, en les décalant les un par rapport aux autres de quelques pixels. On initialise aussi les bar de vie en les remplissant en fonction de la vie des alliés et on les places au dessus de leur allié respectif. On positionne aussi l'ennemi au bon endroit ainsi que sa barre de vie.

-On initialise un sprite d'une cage au cas ou l'ennemi possède un prisonnier. Il sera afficher ou non.

-On initialise ensuite les textes qui seront affichés durant le combat et les options du menu. Notre menu d'action ne possède cependant qu'une seule option viable : attaquer. Les 3 autres n'ont pas été implémenté mais nous les avons laissé pour montrer qu'il est possible d'améliorer le jeux et d'aller plus loin.

-On initialise enfin les positions des items de chaque allié. Ces derniers sont placés au dessus des alliés.

Une fois que tout cela est fait on commence la boucle while qui durera tout le long du combat. A chaque tour on draw la map et tous les personnages ainsi que le prisonnier et la cage si besoin. Les alliés vont devoir choisir une action a tour de rôle. Comme dit précédemment seul l'option d'attaque est possible. Lorsque l'option "attaquer" est choisie une animation d'attaque se déclenche avec la fonction "attaque_animation". Une fois que tous les alliés ont attaqués c'est autour de l'ennemi de riposter sur le dernier allier de la liste (celui qui est draw le plus proche de lui sur la map). On recommence tout cela jusqu'à ce que le combat soit remporter par les alliés ou l'ennemi (qui aura vaincu tous les alliés). Une fois que le combat est finis, s'il s'agit d'une victoire on a une musique de victoire qui est jouée et on retourne sur la map du "Game". Si il y avait un allié emprisonné il est libéré et rejoint l'équipe du joueur.

3 Fonctions Non-Méthodes

3.1 Attaquer_animation

Cette fonction permet d'implémenter l'animation d'une attaque d'un allié vers l'ennemi lors du combat. Elle prend en argument l'allié qui attaque, le joueur, l'ennemi, le sprite de la map de combat, le sprite de la cage.

On commence par faire se déplacer l'allié vers l'ennemi en le rapprochant de 3 pixels par tour de boucle. A chaque tour on draw la map, le joueur, les autres alliés et l'ennemi ainsi que les items et les barres de vie. Une fois que l'allié qui attaque est assez proche de l'ennemi il l'attaque. On a un son d'attaque qui est déclenché et l'ennemi perds un montant de points de vie égal à l'attaque de l'allié. Sa barre de vie est mise à jour. Ensuite l'allié fais le trajet inverse et retourne à sa position initial.

3.2 Attaquer_animation2

Cette fonction permet à l'ennemi d'attaquer un allié précisé en argument. Elle fonctionne exactement de la même manière que la fonction "attaque_animation".

4 Installation

La seule installation qu'il faut effectuer avant de pouvoir compiler le jeu est l'installation de la bibliothèque graphique SFML, en utilisant la commande "sudo apt-get install libsfml-dev" dans l'invite de commande linux.

5 Makefile

Afin de compiler le jeu, il suffit d'entrer la commande "make" dans l'invite de commande linux après s'être placé dans le répertoire courant du projet. L'exécution de cette commande va dans un premier temps compiler tous les fichiers possédant une extension ".cpp" en fichier objets possédant une extension ".o". Elle va ensuite utiliser ces fichiers objets pour créer le fichier exécutable "main" que l'on peut ensuite lancer en utilisant la commande "./main" dans l'invite de commande linux. Il est à noter qu'une fois la commande "./main" exécutée, le jeu pourrait prendre une trentaine de secondes avant de se lancer, car il faut que le programme charge tous les fichiers audio et image que nous utilisons au traver de la bibliothèqu SFML, ce qui peut prendre un peu de temps.
On peut également utiliser la commande "make valgrind" dans l'invite de commande linux afin de compiler le projet comme dans l'opération précédente mais en terminant en utilisant cette fois-ci la commande "valgrind ./main" pour afficher les erreurs valgrind ainsi que les potentielles fuites de mémoire du programme.
Il existe également une commande "make clean" qui permet de supprimer tous les fichiers objets et exécutables du répertoire courant.

6 Erreurs valgrind ?

Sur certaines machines l'exécution de la commande "make valgrind" peut prendre un temps assez conséquent, nous résumerons donc ici les effets de son utilisation. Plusieurs erreurs apparaissent après exécution du programme en utilisant

valgrind, cependant elles semblent provenir de librairies que nous n'utilisons pas directement. Après d'amples recherches sur internet pour tenter de comprendre d'où nos quelques fuites mémoire et erreurs valgrind venaient, il semblerait que ce soient des problèmes liés à l'utilisation de la bibliothèque graphique SFML, et que ce genre de problèmes soient plutôt communs chez leurs utilisateurs.

7 Tests Unitaires

Nous avons également implémenté des tests unitaires afin de tester la majeure partie des méthodes utilisées par nos classes, et que l'on peut retrouver dans le fichier "TestCase.cpp" à l'intérieur du dossier "tests". Un second makefile est situé à l'intérieur de ce dossier. En utilisant ce dossier en tant que répertoire courant, exécuter la commande "make testcase" générera un fichier exécutable nommé "testcase" qui, à son exécution, lancera les tests implémentés dans le fichier "TestCase.cpp". Une commande "make clean" est également utilisable par ce makefile qui supprimera comme le précédent tous les fichiers objets exécutables et objets générés par la commande "make testcase".

8 Comment Jouer

Le jeu démarre par l'apparition du joueur dans la première carte. Le joueur peut de déplacer dans chaque carte en utilisant les flèches directionnelles et consulter son inventaire en appuyant sur la touche "I" (On ne peut pas encore manipuler les objets dans l'inventaire). Lorsque le joueur s'approche trop près d'un ennemi, un combat commence. Le joueur peut alors ordonner à ses alliés d'attaquer l'ennemi en appuyant sur la touche 1. Certains ennemis prennent des alliés en otage représentés par des personnages emprisonnés dans une cage lors de la scène de combat. Lorsqu'un ennemi qui possède un otage est vaincu, il libère son otage qui vient renforcer les rangs des alliés du joueur. Dans la première salle, pour accéder aux salles suivantes, il faut se diriger en bas à droite et aller au delà des deux ennemis sacs poubelle (et suivre le chemin). Le joueur peut alors suivre le sentier en décidant d'aller combattre (ou non) les différents facteurs de pollution de la vie courante selon son opinion sur le sujet. Une fois arrivée dans la dernière salle, un ennemi puissant vous fera face...

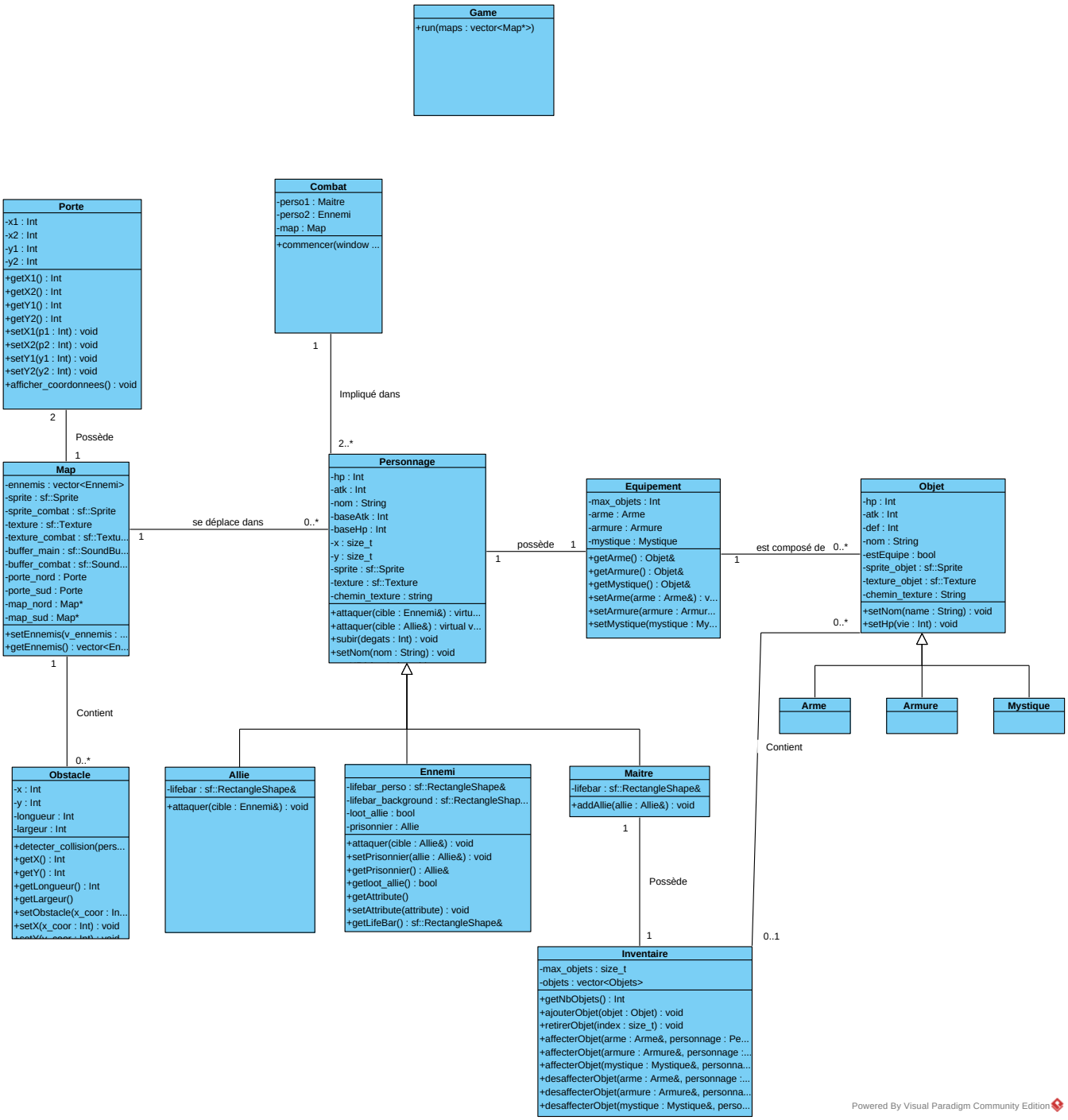
9 Fiertés et Ouverture

Nous sommes assez fiers du rendu du jeu en général, mais nous regrettons de ne pas disposer d'un peu plus de temps pour ajouter quelques détails qui auraient mérité de l'être dans notre version finale pour ajouter plus de consistance dans l'expérience de jeu. Cependant, nous sommes fiers de la classe Combat, qui est notre manager de Combat entre le joueur et un ennemi. Cette idée de transformer le système de combat et de l'incorporer à l'intérieur d'une classe a aidé à rendre la boucle de jeu beaucoup plus simple et nous sommes également

fiers des petits détails sonores ainsi que visuels ajoutés à l'expérience de combat (ainsi que l'expérience sonore hors combat). Les premières fonctionnalités que nous rajouterions avec un peu plus de temps seraient la possibilité de gérer l'inventaire directement depuis le menu d'inventaire (au lieu de juste les afficher), nous mettrions en place un système de loot d'objets pour lequel nous manquions cruellement de temps, et nous améliorerons également l'aspect visuel de nos cartes. Le dernier point auquel nous pensions serait de créer de nouvelles classes qui, tout comme la classe "Combat", serviraient à gérer des parties précises de la boucle de jeu, avec par exemple une classe dont le rôle serait de gérer l'affichage à l'écran, ou encore une classe qui gérerait le déplacement du joueur afin de donner encore plus de clarter à la boucle de jeu principale dans la classe "Game".

10 Diagramme UML

Voir page suivante


Game
+run(maps : vector<Map*>)
Combat
-perso1 : Maitre
-perso2 : Ennemi
-map : Map
+commencer(window ...
Porte
-x1 : Int
-x2 : Int
-y1 : Int
-y2 : Int
+getX1() : Int
+getX2() : Int
+getY1() : Int
+getY2() : Int
+setX1(p1 : Int) : void
+setX2(p2 : Int) : void
+setY1(y1 : Int) : void
+setY2(y2 : Int) : void
+afficher_coordonnees() : void
1
Impliqué dans
2..*
2
Possède
1
Map
-ennemis : vector<Ennemi>
-sprite : sf::Sprite
-sprite_combat : sf::Sprite
-texture : sf::Texture
-texture_combat : sf::Textu...
-buffer_main : sf::SoundBu...
-buffer_combat : sf::Sound...
-porte_nord : Porte
-porte_sud : Porte
-map_nord : Map*
-map_sud : Map*
+setEnnemis(v_ennemis : ...
+getEnnemis() : vector<En...
1
se déplace dans
0..*
1
Personnage
-hp : Int
-atk : Int
-nom : String
-baseAtk : Int
-baseHp : Int
-x : size_t
-y : size_t
-sprite : sf::Sprite
-texture : sf::Texture
-chemin_texture : string
+attaquer(cible : Ennemi&) : virtu...
+attaquer(cible : Allie&) : virtual v...
+subir(degats : Int) : void
+setNom(nom : String) : void
1
possède
1
Equipement
-max_objets : Int
-arme : Arme
-armure : Armure
-mystique : Mystique
+getArme() : Objet&
+getArmure() : Objet&
+getMystique() : Objet&
+setArme(arme : Arme&) : v...
+setArmure(armure : Armur...
+setMystique(mystique : My...
1
est composé de
0..*
Objet
-hp : Int
-atk : Int
-def : Int
-nom : String
-estEquipe : bool
-sprite_objet : sf::Sprite
-texture_objet : sf::Texture
-chemin_texture : String
+setNom(name : String) : void
+setHp(vie : Int) : void
0..*
Arme
Armure
Mystique
Contient
1
Contient
0..*
Obstacle
-x : Int
-y : Int
-longueur : Int
-largeur : Int
+detecter_collision(pers...
+getX() : Int
+getY() : Int
+getLongueur() : Int
+getLargeur()
+setObstacle(x_coor : In...
+setX(x_coor : Int) : void
Allie
-lifebar : sf::RectangleShape&
+attaquer(cible : Ennemi&) : void
Ennemi
-lifebar_perso : sf::RectangleShape&
-lifebar_background : sf::RectangleShap...
-loot_allie : bool
-prisonnier : Allie
+attaquer(cible : Allie&) : void
+setPrisonnier(allie : Allie&) : void
+getPrisonnier() : Allie&
+getloot_allie() : bool
+getAttribute()
+setAttribute(attribute) : void
+getLifeBar() : sf::RectangleShape&
Maitre
-lifebar : sf::RectangleShape&
+addAllie(allie : Allie&) : void
1
Possède
1
0..1
Inventaire
-max_objets : size_t
-objets : vector<Objets>
+getNbObjets() : Int
+ajouterObjet(objet : Objet) : void
+retirerObjet(index : size_t) : void
+affecterObjet(arme : Arme&, personnage : Pe...
+affecterObjet(armure : Armure&, personnage :...
+affecterObjet(mystique : Mystique&, personna...
+desaffecterObjet(arme : Arme&, personnage :...
+desaffecterObjet(armure : Armure&, personna...
+desaffecterObjet(mystique : Mystique&, perso...
Powered By Visual Paradigm Community Edition