Demonti Pierre
Sarrazin Alexandre
Soussan Jimmy

Rapport Projet C++

Première partie du projet :

Lors de la remise du sujet du projet, nous nous sommes réunis pour tout d'abord nous mettre d'accord sur chaque point du sujet et pour que l'on ait tous la même vision de ce que nous devions faire. Ensuite lorsque nous avions repéré les points essentiels du projet, nous avons décidé d'écrire sur papier toutes les classes que nous avons décidé d'utiliser. Dans chacune de ces classes, nous avons évoqué les principales fonctions qui pouvaient s'y trouver, nous avons aussi défini les différents Getters et Setters dont nous aurions besoin et les différents constructeurs de chacune de ces classes.
Cette étape nous a permis d'avoir une vision globale de l'ensemble du projet et de savoir comment nous allions pouvoir commencer a coder.

Le choix de l'interface graphique a été fait au même moment et nous avons décidé d'utiliser l'interface graphique SFML.

Deuxième partie du projet :

Étape **1 :**

Une mise en place des classes principales a été faite dans un premier lieu et un makefile a été créé pour pouvoir nous simplifier l'utilisation de commande lorsque nous faisions des test. C'est-à-dire que nous avons créé la classe de la map avec son constructeur, son destructeur et ses différents getters et setters. Dans cette classe Map, nous avons aussi utilisé un vector d'objets cela nous permet de stocker tous les objets présents sur la carte sous une forme de liste. Grâce à ce vector d'objets, nous avons un accès à chaque objets présents dans la liste et pouvons y faire des modifications sans avoir de contraintes. La Map contient aussi 2 joueurs.
Nous avons aussi dû concevoir une classe Point pour pouvoir avoir accès a des abscisses et des ordonnées dans notre représentation graphique.

La première étape de rédaction du code a été d'un côté de créer la map et d'un autre côté de créer l'éditeur de map tout en s'occupant de l'affichage graphique de nos différents éléments.
Pour la map, notre premier objectif était d'y mettre des objets, nous avons au préalable rempli une liste d'objet et avons testé si la map contenait bien la liste et qu'il y avait le bon nombre d'objets à l'intérieur.

Le test a été le suivant :

```
Map M = Map(10, 10, Point(5,5));  → Ici création d'une map
M.init_liste() ; → la fonction init_liste va créer une liste contenant des objets
cout<<M.getTailleListe()<<endl; Ce qui nous permet d'afficher la taille de notre liste
```

La classe Objet contient tous les paramètres nécessaires qui sont la force, la position et son Id, l'Id permettant le choix de plusieurs objets. Dès la classe d'objets terminée, nous avons tester que nous pouvions bien supprimer les objets présents sur la map ou bien leur faire diminuer de force.

La fonction de test :

```
Objet O = Objet(Point(200,200),id::Rocher,10);
Objet A = Objet(Point(100,100),id::Arbre,10);
A.takeDamage(10);
this→list.push_back(O);
this→list.push_back(A);
```

Après ce test qui comptait d'autres parties de fonction, l'objet A n'avait plus de vie. Et n'apparaissait plus.

En parallèle de cette partie, nous avons travaillé sur l'éditeur de texte. Pour l'éditeur de texte, il a fallu que nous commencions la partie graphique de la map, pour pouvoir sélectionner des objets et les placer sur la map. Afin d'enregistrer les positions et les id des objets sélectionnés.
Deux fonctions nous ont été utiles l'ajout d'objets dans une liste qui appartient à la map et la suppression des objets si la personne qui crée la map souhaitent revenir en arrière lors de la création de celle-ci. Cette dernière fonction parcours la liste d'objet et si l'objet appartenant à la liste est sélectionné elle le supprime de la liste.

Pour sauvegarder la map, nous avons utiliser ofstream. Nous sauvegardons les données propres à la création de la map puis lorsque ses données sont sauvegardées. Nous parcourons la liste d'objet et insérons les données de chaque objet ligne par ligne.

Étape 2 :

Après toute la partie concernant la réalisation du monde virtuel, nous avons travaillé sur la classe Personnage.
Cette dernière contient un constructeur avec toutes les caractéristiques nécessaires au personnage et des getters et des setters pour qu'on puisse avoir accès au différentes caractéristiques du personnage. Cette classe contient aussi, une fonction qui va nous permettre de savoir si le personnage a été touché par un tir et une fonction permettant au personnage de tirer.

Pour la sélection du mode jeu, nous avons créé plusieurs gestions.
La première gestion est l'ordinateur contre l'humain, dans cette gestion nous utilisons aussi une gestion de la map spécifique. Dans cette gestion, quand le joueur va choisir une action spécifique, le code va tester si il est possible de réaliser l'action demander par le joueur et dès que le joueur n'a plus de pas la gestion de l'ordinateur va être lancer. La partie s'arrêtera dès que l'un des deux sera mort.
La seconde gestion est ordinateur contre ordinateur, cette gestion va lancer une partie avec une map et lancer la gestion de l'ordinateur. La gestion de l'ordinateur va exécuter des déplacements aléatoires.
La dernière gestion est le joueur contre un autre joueur en tour par tour. La console va attendre une action de la part du premier joueur et lorsque ce joueur ci n'a plus de déplacement possible, le joueur va pouvoir tirer en appuyant sur espace ou bien passer son tour en appuyant sur entrée.

Dans chacune des gestions, la collision du personnage avec les différents éléments est testé à chaque déplacement.

Exemple :

```
for(Objet &O: M.getliste()){
        if (collision(P,O,M) == true){
                test = false ;
        }
        else if (collision(P,O,M) == false){
                test = test && true ;
        }
}
```

Cette exemple vérifie si il y a une collision ou non avec chaque objet présents dans la liste de la map. Si il y a une collision, le personnage ne va rien faire sinon il va pouvoir se déplacer.

Puis nous avons appréhender le tir, le tir doit partir du personnage et aller dans une direction en s'arrêtant en fonction de la portée de l'arme que possède le personnage ou bien il s'arrête lorsqu'il rencontre un objet. Lorsque le tir va rencontrer un obstacle, cette obstacle va prendre des dégâts et le tir doit s'arrêter. Nous avons aussi dû bien réfléchir à l'orientation du tir car le tir doit pouvoir partir en diagonale. Nous avons dû trouver une formule car lors de notre première implémentation du tir nous le faisions partir soit en haut, en bas, à droite ou à gauche en fonction de l'orientation du personnage. Selon le choix de du joueur, l'angle de tir se trouvera entre 0 et 90 degrés et la direction sera alors calculée d'après la formule ci-dessous :

float x = getPositionActuelle().getAbs()+sin(orientation*3,141592/180);

et puis on change la position avec :

```
setPositionActuelle(Point(x,a*(x-getPositionDepart().getAbs())
                        +getPositionDepart(),getOrd())));
```

Lorsque le tir se déplaçait correctement, nous avons effectué la collision tir/objet avec le perte de vie des objets en fonction de la puissance de tir. Et si l'objet rencontré n'a plus de vie, il disparaît de la map et le joueur peut alors se mettre sur la place de l'objet.

Etape 3 :

La dernière partie consiste à assembler tout ce que nous avons fait pour avoir un jeu complet avec une interface graphique.

Nous avons fait un menu d'accueil dans lequel le joueur peut décider si il va créer un terrain de jeu, ou bien jouer une partie ou tout simplement quitter le jeu.
Si le joueur décide de créer sa propre map, on va lui demander d'écrire la taille de la map, ensuite la map va s'afficher et le joueur va pouvoir placer différents objets sur la map avec des tailles différentes. Pour sauvegarder sa création, il devra appuyer sur la touche 'S'.
Si le joueur souhaite jouer, il va arriver sur un écran de sélection pour savoir contre qui il va jouer.

Après ce choix il devra décider sur quel map il souhaite jouer et dès que son choix sera fait la partie se lancera. Pour s'orienter, il faudra utiliser les flèches de droite et de gauche, et pour avancer les flèches du haut et du bas. Lorsqu'il aura atteint son nombre de pas maximum, il pourra décider de tirer avec la touche espace, mais avant de tirer il pourra choisir l'orientation de son tir, ou bien de

passer son tour avec la touche entrée. La partie se terminera quand un des deux joueurs aura touché l'autre.

Nous avons aussi mis en place le système de visibilité sur la map, c'est-à-dire que si le joueur se cache derrière un objet, l'ordinateur ou l'autre joueur ne pourra pas le voir et vice-versa. Mais si il n'y a pas d'obstacles entre les deux, ils se verront.

Cette fonction est la fonction vision, elle va prendre en entrée les deux personnages présents sur la map et la map en elle-même. Dans cette fonction, nous faisons appel à la fonction collision_vision qui va renvoyer vrai si il y a un objet entre les deux personnages.

Dernière partie du projet :

Dans ce projet, nous avons fait un mélange entre la programmation orientée objet et la programmation fonctionnelle.

C'est-à-dire que nos classes principales sont des classes orientées objets par exemple la classe map où il y a un constructeur et des getters et des setters pour pouvoir accéder aux différentes variables appartenant à la map. Dans nos classes, nous avons besoin de pouvoir créer des objets et de les modifier tout au long du jeu, c'est pour cela que la programmation orientée objet est la plus adaptée. Mais pour les modes de jeu, nous avons opté sur de la programmation fonctionnelle avec des procédures de type langage C.
Ce mélange entre ses deux types de programmation est ce qui est essentiel dans le langage c++.

De plus, nous avons laissé des «couts» pour pourvoir justifier la partie sans graphique. Et dans le main, nous laissons des tests que nous avions au tout début pour montrer l'évolution du projet.

Pour l'affichage, nous avons décidé de la faire sous forme de programmation fonctionnelle, c'est-à-dire qu'il fonctionne comme un affichage qu'on aurait pu écrire dans un code en C. Nous avons des fonctions basiques qui affiche chaque élément et une fonction générale nous permettant d'afficher la map entière.

Nous avons pris quelques décisions comme le fait que le joueur puisse apparaître dans un objet et pour qu'il puisse se déplacer il va devoir choisir de tirer.

Descriptif des touches :

Dans le menu :

-Flèches directionnelles haut bas
-Entrer pour valider le choix

Dans l'éditeur :

- 'S' pour sauvegarder
- 'Z' pour effacer l'objet
- Flèche du haut pour augmenter la vie de l'objet
- Flèche du bas pour diminuer la vie de l'objet
- Clic gauche sur l'objet pour le sélectionner et clic gauche pour le placer sur la map

Dans la sélection de niveaux :

- Flèche du haut et du bas pour choisir la map et entrée pour sélectionner

Dans le jeu :

-Flèche du haut pour avancer
-Flèche de droite et de gauche pour s'orienter
-Entrer pour passer son tour
-Espace pour tirer