MOURGUES Paul

QUINTY Julie

B3 - Game Programming - Ynov

SHOOTER - SFML

# Sommaire

1. **Explication générale du jeu - Features fonctionnelles**
2. **Organisation du projet**
3. **Explications des méthodes et classes**
4. **Difficultés rencontrées**
5. **Conclusion**

# Explication générale du jeu - Features fonctionnelles

Le projet initial était de faire un Shooter en SFML. Actuellement, voici les features fonctionnelles du jeu et celles que l’on souhaitait ajouter si le temps nous le permettait.

Fonctionnel :

* Spawn de la map à partir d’un fichier .txt et d’une tilemap.
* Spawn du joueur
* Spawn des ennemis
* Affichage d’un menu avec les règles du jeu
* Déplacement du joueur
* Déplacement automatique des ennemis
* Collision du joueur avec les murs présent sur la map
* Tir des ennemis
* Tir du Joueur
* Fin de partie

Optionnel :

* Gestion des PV du joueur
* Possibilité de changer l’arme du joueur
* Pathfinding des ennemis
* Croissance de la difficulté

# 

# Organisation du projet

L’objectif pour nous était de réaliser un jeu vidéo de type shooter en 15 semaines à 2 avec la SFML. Nous nous sommes servi de l’IDE Qt et de GitHub pour la sauvegarde du projet.

Pour répartir les différentes tâches du projet, nous avons scindé la plupart du travail en deux et réunit régulièrement nos versions afin d’éviter les grosses incohérences sur la fin de projet.

Paul s’est occupé des classes Menu, Renderfield, Player

Julie des classes Assets, Bullet, Enemy.

Les classes Game & Field ont été faites à deux.

On a commencé par setup le menu et le spawn du terrain. Puis sur ce terrain, nous avons fait spawn les ennemis et le joueur. Et enfin, nous avons ajouté les tirs, la gestion de collision et les conditions de fin.

L’intérêt du projet était également de perfectionner notre maîtrise du cpp comme se fut le cas avec l’utilisation de design pattern comme *les singleton,* la limitation de l’utilisation de static et une gestion des classe la plus cohérente.

## Classe Assets

Cette classe contient toutes les méthodes liées à la gestion des Sprites. Elle se devait d’avoir toutes ses méthodes en virtual puisqu’elle sera utilisée dans beaucoup d’autres classes.

Elle contient ses constructeurs, destructeurs, getters, setters. La méthode **turnAsset** permet de faire tourner les ennemis dans le sens de leur marche durant leur ronde.

## Classe Bullet

Cette classe contient toutes les méthodes liées à la gestion et au mouvement des balles. En dehors de son constructeur / destructeur, nous trouverons ici :

* **drawBullet** : Cette méthode permet de construire visuellement la balle
* **moveBullet** : Cette méthode permet de modifier en continue la position de la balle suivant un pas donné en paramètre.

## Classe Enemy

Cette classe contient les méthodes de création et de gestion d’ennemis. Elle contient également :

* **EnemyDraw** : Cette méthode permet de construire visuellement l’ennemi
* **SpawnShot** : Cette méthode permet d’initialiser la balle à la position de l’ennemi.

## Classe Field

Cette classe permet l’affichage du terrain. Elle permet également l’initialisation des ennemis sur le map ainsi que leurs mouvements grâce à :

* **getTilesFromFile** : Cette méthode permet de récupérer la map depuis un fichier texte et inclut chaque tuile du terrain dans un vecteur.
* **checkMap** : Permet de poser le bon sprite sur le terrain en fonction de la lettre présente dans le fichier texte.
* **PutEnemies** : Permet de faire spawn les ennmies sur la map et de les inclure dans un vecteur d’Ennemis.
* **UpdateEnemies** : Permet de faire bouger les ennemis sur une plage donnée avec un pas donné. Elle permet également de les faire tirer toutes les 7 secondes.
* **enemiesShoot** : Permet de faire avancer les balles.

## Classe Renderfield

Cette classe permet de lire le fichier .txt et de déterminer les sprites qui composeront la map. Elle est appelée avant la class Field afin de lui préciser où elle doit afficher tel sprite grâce à la méthode **initSpriteOnMap.**

## Classe Game

La class Game va avoir toutes les méthodes qui vont faire la mécanique de jeu. Elle initialise la fenêtre et fait fonctionner le jeu (méthode : **run**), récupère les commandes faites par le joueur (méthode : **processEvents** & **handlePlayerInput)**, gère l’update (méthode : **update**) et l’affichage des acteurs du jeu (méthode : **render**).

## Classe Menu

Comme son nom l’indique, nous allons retrouver ici toutes les méthodes qui nous permettent de gérer et d’afficher le menu avant de lancer le jeu. Entre autres, on peut y inscrire son nom consulter les scores, les touche etc… a therme la classe menu aurait dû servir de hub pour le jeu.

## Classe Player

La clases Player sera notre dernière classe au sein de cette documentation. Elle va regrouper toutes les méthodes liées au joueur telle que sa construction, son affichage, sa gestion des mouvements, les changements de scène liés à son déplacement ainsi que le lancement de projectiles.

# Objectifs & Difficultés rencontrées

Julie :

Dans mes objectifs figurait l’amélioration de mes acquis mais surtout pouvoir combler mes lacunes restantes. C’est donc naturellement que parmi les difficultés rencontrées les plus bloquantes, la gestion de Sprite multiples a été le plus difficile à comprendre. N’ayant que très peu de connaissances en c++ au début du projet, j’ai dû faire face à l’apprentissage des pointeurs en plus de l’avancée du projet. Ca a également été un frein dans la productivité et la progression de certaines tâches, puisque j’ai dû me former pour arriver à mes fins.

Paul :

Pour une personne pratiquant le c++ depuis plus longtemps, j’été surtout intéressé par essayer de rendre un code plus lisible et reprenable par une personne tierce. Je me suis essayer au design pattern tout en améliorant mes connaissance des pointeur, des classes et l’appel de variable extérieures.

Les problèmes que j’ai rencontres se sont surtout situé en fin de projet, en effet je me suis retrouver avec une gestion des scènes/salles fonctionnels sur le papier mais des lacunes dans le codages des premières fonction m’as empêché de régler totalement le problème, de la même manière la collision entre les balles et les entités sur le terrain ont été un gros défis et ne sont pas encore totalement fonctionnels, c’est un point que j’aurais aimé améliorer. Enfin la gestion du temps entre les cours et la recherche de stage/alternance on été compliqué et j’espère avoir appris à gérer mon temps plus efficacement.

# Conclusion

Le projet était tout de même très enrichissant. Malgré des lacunes au départ, nous estimons avoir progressé sur la compréhension du c++ et sa manipulation en global.

Il reste cependant beaucoup de choses à apprendre/perfectionner et ce projet nous permettra sans aucun doute d’avoir une nouvelle gestion de nos projets futurs.