Université Gustave Eiffel

BERNIER Aurélien
BRUNEL Matthias

Rapport

Projet de Programmation C : Tower Defense

I. Introduction

Objectifs

Le objectif de ce projet était de développer une application graphique en C un jeu de *tower defense*. Il s'agit typiquement d'un jeu d'action-stratégie, dans lequel le joueur doit maintenir un système de tours pour protéger sa base contre des vagues d'ennemis

L'application est basée sur la libMLV pour l'affichage graphique.

Répartition des tâches

Nous avons commencé par modulariser le projet ensemble et Matthias a écrit le makefile.

Aurélien a réalisé la génération du chemin (Path), les vagues de monstres (Monster, MonsterList) et l'affichage graphique (Engine).

Matthias a réalisé les tours (Tower, TowerArray), gemmes (Gem, GemList), les tirs (Shot, ShotList) et la mana (ManaTank). Il a aussi écrit la boucle principale du jeu (Game).

Les fichiers dans le dossier « *utils* » ont été écrit suivant les besoins de chacun.

II. Mise en œuvre

Modularisation

On a choisi de séparer chaque objet dans un module différent, ainsi chaque fichier C s'occupe d'un élément distinct du jeu. Les modules sont classés dans les dossiers ainsi :

- game : Contient les objets du jeu tels que les gemmes, monstres, tours et tirs ainsi que les listes permettant de gérer l'ensemble de ces éléments.
- graphics : Contient le module qui gère l'affichage du jeu
- utils : Contient les modules utilitaires utilisés par les autres fichiers, tel que Coord qui représente le système de coordonnées utilisés dans tout le projet.

Game

Le jeu est géré par le fichier game.c qui contient notamment les fonction game_loop et game_tick pour gérer la boucle de jeu principale. La structure Game nous permet de stocker tous les éléments contenus dans le jeu tels que les listes de monstres, gemmes, tours et tirs. Ce fichier contient aussi les fonctions permettant d'agir sur le jeu en améliorant la réserve de mana, construire une tour, générer une gemme ou les fusionner.

Le ManaTank représente la réserve de mana avec son niveau, sa capacité et son contenu et possède une fonction pour être initialisé, augmenté en niveau et pour gérer les différentes augmentations ou diminutions de mana, tel que mt_kill_monster pour gagner du mana à la mort d'un monstre ou mt_buy_tower pour perdre du mana lors d'achat d'une tour.

La GemList est une liste chainée de Gem. Les Gem peuvent être créés, fusionné, copiés et ajouté/supprimés de la liste. Elles possèdent un niveau, une couleur et un type (RED, GREEN, BLUE si elle est pure, sinon MIXED).

La MonsterList est une liste chainée de Monster. Ce fichier permet de générer des vagues de monstre tels que décris dans le sujet. Les monstres étants représentés par leurs points de vies, vitesse, couleur, coordonnées et les éventuels résidus ou debuff à leur appliquer.

Le Path est un tableau de cellules (Cell) ayants une coordonnée, un booléen isPath et une direction dir (North, South, East, West). Le path contient aussi les coordonnées du nid de monstre et de la base, ainsi que sa longueur et nombre de tournant. Le chemin est généré par la fonction path_gen en utilisant l'algorithme fournis dans le sujet.

Le TowerArray est un tableau de Tower. Les Tower possèdent une coordonnée et un pointeur vers la gemme contenue dans la tour (peu être null). Les tours peuvent tirer et on peut y ajouter ou retirer une gemme. Les tours ne peuvent pas être supprimés une fois placés.

La ShotList est une liste chainée de Shot qui permet de stocker les projectiles tirés par les tours. Ces projectiles (Shot) possèdent une coordonnée, un pointeur sur la gemme dont il provient et un pointeur vers le monstre qui est la cible. Ce fichier s'occupe de gérer les effets à l'impact du tir en fonctions de la gem dont il provient. Pour éviter que le pointeur devienne null pendant le trajet du tir, on utilise la fonction de copie des gemmes.

Enfin le Game contient 3 variables pour gérer les actions en jeu. La gemme sélectionnée par le joueur lorsqu'il clique dessus pour sauvegarder celle qu'il souhaite fusionner. Un booléen qui mémorise si le bouton pour placer une tour à été actionner. Et enfin la valeur de temps au début de la partie.

Graphics

La partie graphique contient un fichier et utilise 3 fonctions. Une fonction pour initialiser la fenêtre de jeu et le fond, une fonction pour afficher les éléments interactifs et une fonction pour gérer les event (cliques souris).

Utils

Le fichier « constants.h » stocke toutes les constantes utilisées dans le programme.

Coord permet de gérer des coordonnées à 2 dimensions et de faire des opérations simples dessus. FCoord est son équivalent en Flottant au lieu d'Integer.

Le fichier « fatal.h » gère les erreurs du jeu.

Le fichier « time.h » gère le temps pour le jeu.

III. Utilisation

Compilation

- make : Compile le projet, peux ensuite être exécuté via la commande « ./tower_defense »
- make clean : Supprime les objets
- make run : Compile et exécute le projet

Jeu

La mana maximum et restante est affiché en haut a gauche de l'écran, en haut à droite du terrain se trouve le numéro de la vague actuelle.

A droite se trouve l'inventaire des gemmes, les gemmes sont des cercles avec un chiffre représentant leur niveau. Cliquer sur une gemme la sélectionne, cliquer sur une tour la met dans la tour, cliquer sur une autre gemme les fusionnes, cliquer sur une tour contenant une gemme de niveau différent remplace la gemme présente dans la tour.

A droite on peut voir 4 boutons :

- Buy Gem : Achète une gemme de niveau 0 et la place dans l'inventaire
- Buy Tower : Cliquez ensuite sur le terrain pour acheter et placer une tour
- Level Up : Augmente le niveau de la réserve de mana
- Next Wave : Appel la vague de monstre suivante

IV. Conclusion

Difficultés

Nos principales difficultés ont été les déplacements des monstres et des tirs ainsi que certains effets à l'impact des tirs. Et surtout le fait supprimer les tirs qui ciblent un ennemi qui est mort entre temps et actualiser toutes les références à cet ennemi correctement lorsqu'il est supprimé pour éviter des segfault.