Kévin BENOIT
Aurélien CORBLIN

Université Gustave Eiffel

Rapport : projet Tower Defense.

PLAN DU RAPPORT :

INTRODUCTION :

CORPS DU RAPPORT:

COMPILATION :

BUG :

CONCLUSION :

INTRODUCTION :

Dans ce projet il était question de coder, un jeu de Tower Defense en C. Pour y parvenir nous avons modularisé notre code de la sorte :

- Mana
- Gem
- Tower
- Monster
- Path
- Deplacement
- Damage
- Display

CORPS DU RAPPORT :

Ce rapport fait office de rapport technique et d'un rapport normal, car ces deux notions étant assez similaire vous les trouverez tout les deux ici représentés

Dans le module Mana, nous avons effectué la gestion du mana, que ce soit quand on gagne du mana, quand on doit payer avec du mana, ou même quand un monstre passe par le camp du joueur, et on a géré ici le Game Over. Pour cela nous avons créer la structure Mana :

```
typedef struct{
int level;
int qty;
int max;
}Mana;
```

Dans le module Gem, nous avons effectué l'initialisation des gemmes ainsi que la gestion de l'inventaire des gemmes qui comporte son initialisation, l'ajout d'une gemme au niveau souhaitée (forcément pure, car on ne peut acheter que des gemmes pures), la fusion de deux gemmes d'un même niveau, et donc la réorganisation de l'inventaire de gemmes. (suppression et ajout). Pour cela nous avons créer 2 structures :

```
typedef struct{
int color; // 0 - 359
int level;
}Gem;

typedef struct{
Gem inventory[24];
int nb;
}Inventory;
```

Dans le module Tower, nous avons géré l'initialisation d'une tour : sa position, sa gemme, et un entier timeToShoot pour les tirs. Dans Tower initTower(Position pos), une astuce à été

d'initialiser la teinte de la gemme à -1, dans le main on se servira de cela pour effectuer des conditions sur le placement des gemmes sur les tours. Ensuite nous avons créer une structure Towers, qui va gérer toutes les tours présentes sur la grille. Pour cela nous avons créer deux structures :

```
typedef struct{
Position p;
Gem g;
int time_to_shoot;
}Tower;
```

```
typedef struct{
Tower* tab;
int number;
int size;
int priceToPay;
}Towers;
```

Dans le module Monster nous avons effectué la gestion des Monstre, leur initialisation, ainsi que la création d'une vague de monstre, et d'un tableau de vague.
cette fonction :

```
int update_monsters(Waves* waves, Segments* segments, Mana* mana)
```

gère les effets appliquées aux monstres (éléments résiduels temporels), et les mouvements des monstres. Par ailleurs, les monstres ne se déplacent pas tous de la même vitesse, afin que chaque monstre reste bien centrés sur chaque milieu de case il était important de bien gérer leur déplacement, exemple :

un monstre se déplace horizontalement avec une vitesse de 1.5cases/secondes or il se trouve à 1 case du prochain virage, pour que le monstre ne se trouve pas décalé du centre on vérifie que la distance qu'il parcourt en une seconde est inférieure à la distance entre lui et le prochain virage, si c'est le cas le monstre avance normalement on incrémente que les x (rappel : le monstre ici se déplace horizontalement), si ce n'est pas le cas on incrémente les x de la distance entre le monstre et le prochain virage (donc 1) et on incrémente/décrémente (selon la direction) les y par l'excédent (distance parcourue en 1 sec – distance monstre virage) .

Pour cela nous avons utilisé ces structures :

```
typedef struct {
int has_effect; // 0 : no effect , 1 : has effect
int time_left;
int interval;
int damage;
double slow_down;
} Effect;
```

```
typedef enum {
NORMAL,
FOULE,
```

```
AGILE,
BOSS
} Wave_Name;

typedef struct {
double speed;
int hp;
int color; // 0-359
int elementary_residue; // 1 = pyro, 2 = dendro, 3 = hydro
Effect effect;
int dont_move; // special effect, its the time before monster can move
double pos_x;
double pos_y;
int current_segment;
} Monster;

typedef struct {
Wave_Name name;
Monster monsters[24];
int size;
int spawn;
int dead;
} Wave;

typedef struct {
Wave tab[MAX_WAVES]; // limiter a 50 en meme temp sur la grille
int nb_current_waves;
int nb_dead_waves;
int nb_total_waves;
} Waves;
```

Dans le module Path, c'est la que nous avons créé le chemin que les monstres vont parcourir entre leur nid et le camp du joueur, en utilisant les formules données (distance de manhattan) et l'algorithme décrit dans le sujet. Pour cela nous avons utilisé ces structures de données :

```
typedef enum {
NORTH,
EAST,
SOUTH,
WEST
} Dir;

typedef struct {
int line;
int col;
} Position;

typedef struct {
Position path[LINE * COL];
int size;
} Grid;
```

Dans le module Deplacement, nous avons choisi la façon dont les monstres allaient se déplacer sur le path. Pour cela nous avons choisi de représenter le Path en un tableau de Segment, la fonction qui initialise le tableau de Segments `Segments init_segments(Grid grid);` va parcourir le path, et pour chaque virage rencontré va ajouter un segment au tableau. Ainsi quand un monstre va se déplacer il n'aura plus qu'a se déplacer de Segments en segments. Pour cela nous avons utilisé ces structures :

```
typedef struct{
Position deb;
Position fin;
}Segment;
```

```
typedef struct{
Segment tab[30]; // min 6 ou 7
int nb_seg;
}Segments;
```

Dans le module Damage, c'est ici que nous avons géré tout ce qui touche aux dégats effectué par une gemme à un monstre et ce qu'elle que soit son type, on gère la position des tirs, les effets. Pour cela nous avons utilisés ces structures :

```
typedef struct {
int wave_targeted;
int monster_targeted; // index of targeted monster in wave
int color; // color of the gem at the origin of the shot
int niv_gem; // level of the gem at the origin of the shot
double pos_x;
double pos_y;
} shot;
```

```
typedef struct {
shot* tab;
int size;
int nb_shots;
} Shots;
```

Dans le module Display c'est la ou nous avons géré l'affichage graphique, ici nous avons géré l'affichage du path, de la jauge de mana, l'inventaire des gemmes, l'affichage des tours, des monstres, etc ...

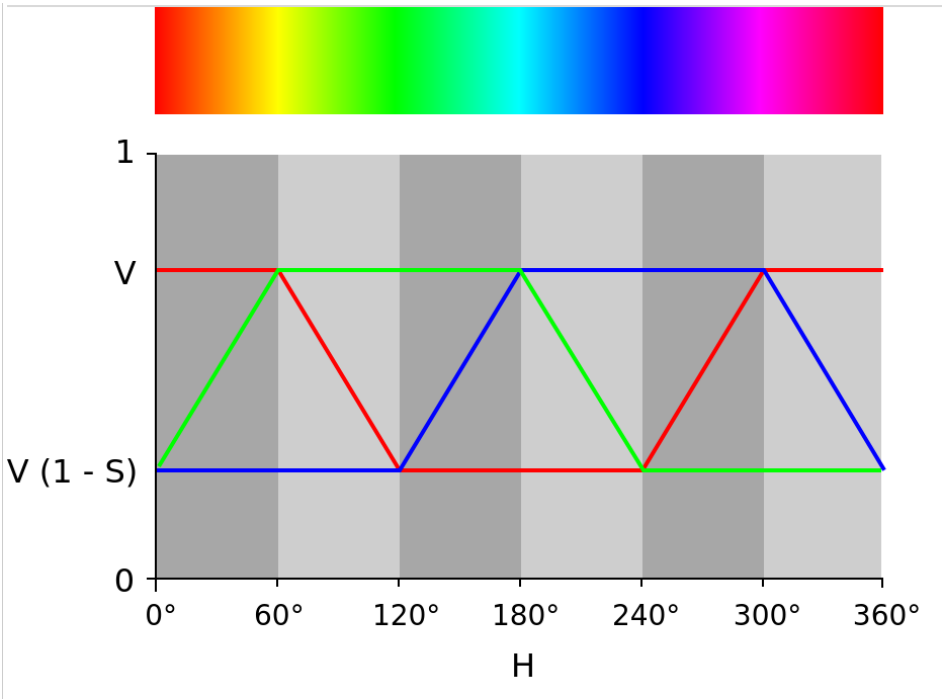
Explication de la fonction :

```
MLV_Color nuanceColorRedToGreen(int quantity_mana, int max_mana);
```

Ici dans cette fonction, ainsi que dans celle ci :

```
MLV_Color nuanceColor(int teinte);
```

on se base sur ce graphique rgb :



nuanceColorRedToGreen va servir pour afficher la jauge de mana et les hp des monstres, ainsi plus on a de mana plus la jauge est verte, moins on a de mana plus la jauge est rouge cela offre un coté plus intuitif au jeu et on se représente mieux l'information car le mana représente un moyen de paiement mais aussi notre barre de vie.

NuanceColorRedToGreen représente ce graphique de 0 à 120.
nuanceColor représente ce graphique de 0 à 360. utilisé pour afficher chaque teinte de chaque gemmes.

Par ailleurs dans notre projet nous avons choisi d'afficher la teinte de la gemme pure pas selon leur type (Pyro Dendro Hydro) donc un simple affichage RGB mais selon leur teinte, cela peut nous porter préjudice, car cela peut porter à confusion sur quelle type chaque gemme comporte. Pour palier à cela, quand on séléctionne une gemme pure (et uniquement) cela affiche son type en dessous de la grille.

Autre fonction intéressante, pour l'affichage du path, on parcours le path 1 et 1 seule fois, sans faire de if car on sait déjà que la premiere case d'indice 0 sera la case des monstres, le reste sera les cases blanches, et la case d'indice taille – 1 sera la case du camp du joueur.

Par Ailleurs pour ce projet nous utilisons une Fonte libre de droit nommée « Orbitron » https://fonts.google.com/specimen/Orbitron , pourquoi choisir d'utiliser une fonte ? Alors outre le fait que c'est joli, etc … c'est libMLV qui nous a contraint a faire cela, car sur libMLV la fonte par défaut s'affiche en très petit, pas très ergonomique.

Or, quand on affiche une fonte, une fois que la fonte est crée on ne peut modifier sa taille, car une font est définie par le path du fichier .ttf qui la caractérise et sa taille.

Donc il faut redéfinir une fonte avec une taille plus grande.

COMPILATION :

Pour compiler ce projet il faut dans le terminal taper la commande :

make

Puis taper la commande :

./projet

MANUEL UTILISATEUR :

après avoir lancé le programme :

pour lancer la première vague taper sur la touche espace.

Pour upgrade la jauge de mana cliquez sur upgrade, si cela s'affiche en rouge vous ne pouvez pas augmenter la réserve, si cela s'affiche en vert vous pouvez l'augmenter.

Pour créer une nouvelle gemme : cliquez sur New Gem, si cela s'affiche en rouge vous ne pouvez pas acheter la gemme, si cela s'affiche en vert vous pouvez l'acheter.

Pour modifier le niveau de la gemme vous pouvez cliquez sur le '+' ou le '-'.

Pour ajouter une gemme sur une tour vous sélectionner la gemme de votre choix dans l'inventaire de gemme et vous cliquer sur une tour. Faites attention quand vous remplacez une gemme, vous perdrez la gemme présente sur la tour !

Pour aller à la prochaine vague, il faut s'assurer d'avoir abattu la dernière vague à avoir spawn. Puis vous pouvez taper sur la touche espace.

Pour sortir de la fenêtre de Game Over tapez sur n'importe quelle touche du clavier.

BUG :

Pendant ce projet on a rencontrés plusieurs bug :

- quand on tuait la vague 2 avant la vague 1, la vague 1 disparaissait aussi. Cela était du à une faute de frappe dans le damage.c a la ligne 89, on avait mis l'indice i au lieu de la vague visée...

- double free or corruption. : quand on sort du programme avec échapp.

- realloc invalid old size, aborted corruption : ce problème arrivait de manière aléatoire, le jeu était lancé et d'un coup ça plante. (le seul réalloc qui est fait, est fait dans l'ajout d'un nouveau tir et qu'il utilise un pointeur alloué par un malloc et qui n'a jamais été free entre le malloc et le réalloc et les realloc entre eux) donc on ne comprend pas pourquoi ...

CONCLUSION :

Ce projet nous a appris à respecter un cahier des charges afin de mener à bien un projet de type jeu chose que l'on avait fait que en python il y a longtemps (ricosheep) . Cela nous a aussi permis de bien réviser.