Nom 1 : TEKPA
Prenom : Max
Groupe : 1

———

Nom 2 : Tran
Prenom : Thien-Loc
Groupe : 1

Projet de Programmation C : Tower Defense

L3 informatique, Université Gustave Eiffel

Rapport : Tower Defense

1. Modularisation

Nous avons décidé de découper le projet en 7 modules plus le main.c

- Terrain.h : Ce module définit les types et les fonctions liés à la gestion de la grille et la génération du chemin des monstres. Il contient notamment la définition du type enum «caseType» qui représente l'ensemble des cases possibles sur la grille du jeu et du type «Card» qui lui représente les direction possible du chemin et servira également au déplacement des monstres. Le module propose des fonctions pour initialiser la grille du jeu, générer le chemin des monstres par rapport à une direction données, initialiser le nid des monstres et la case joueur (case à défendre).

- Vague.h : Le module vague définit les types et les fonctions liés à la gestion des vagues de monstres dans le jeu. Il contient notamment la définition du type enum TypeVague qui représente les types de vagues possibles, et du type Effect qui représente les effets possibles sur les monstres, aussi des structures comme Monstre qui représente un monstre avec ses attributs comme (sa position, vitesse, point de vie…), Vague qui contient une liste de monstre, leur nombre et enfin TableVague qui représente l'ensemble des vagues de la partie. Le module propose des fonctions pour initialiser les monstres, les vagues, et le tableau de vagues. Il fournit également des fonctions pour déterminer la direction initiale d'un monstre lorsqu'il sort du nid, changer la direction d'un monstre en fonction des cases occupées autour de lui, déplacer un monstre en fonction de sa vitesse et de sa direction actuelles, réarranger la liste de monstres, supprimer un monstre de la liste de monstres de la vague et gagner du mana, réinitialiser la vitesse d'un monstre si les conditions sont remplies, activer les effets sur le monstre, et déplacer tous les monstres dans toutes les vagues. Il contient également une fonction pour gérer l'ensemble des vagues.

- Gemmes.h : Gemme contient l'ensemble les types et les fonctions liés à la gestion des gemmes dans le jeu. Il contient notamment la définition du type enum Type_gem qui représente les types de gemmes possibles, et du type Element qui représente les éléments (PYRO, HYDRO, DENDRO, NONE) de gemmes. Le module propose des fonctions pour créer une nouvelle gemme, réarranger la liste des gemmes, et fusionner deux gemmes pour augmenter le niveau. Il contient également une structure Gem pour représenter une gemme,

qui comprend le type de la gemme, l'élément de la gemme, la teinte, le niveau, et un champ emprise pour enregistrer si la gemme a été sélectionnée.

- Mana.h : Définit la structure et les fonctions liés à la gestion de la réserve de mana dans le jeu. La seule structure du module nommée Reserve représente une réserve de mana, avec un maximum de mana obtenable, la quantité de mana actuelle, et le niveau actuel de la réserve de mana. Le module propose des fonctions pour initialiser une nouvelle réserve de mana, augmenter le niveau de la réserve de mana, augmenter la quantité de mana du joueur en fonction de la vie du monstre tué, et réduire la quantité de mana de la réserve afin de bannir un monstre.

- Graphique.h : Ce module définit les fonctions liées à l'affichage graphique du jeu. Il contient des fonctions pour afficher le mana, les tours, les projectiles, les gemmes, le déplacement d'un monstre, l'ensemble des vagues de monstres, le terrain du jeu, le chemin des monstres, et tout le jeu (terrain, monstres, tours…). Il propose également une fonction pour obtenir la couleur d'une teinte selon son numéro. Le module inclut plusieurs autres modules tels que "gemmes.h", "mana.h", "terrain.h", "game.h", et "vague.h" pour utiliser leurs structures et fonctions respectives.
- Moteur.h : Ce module définit les fonctions liées à la gestion des événements et des actions du jeu. Il contient notamment des fonctions pour gérer les événements du clavier et de la souris pendant le jeu, gérer les actions du joueur pendant le jeu, et la boucle principale du jeu. Le module inclut plusieurs autres modules tels que "graphique.h", "terrain.h", "vague.h", "gemmes.h", "mana.h", et "game.h" pour utiliser leurs structures et fonctions respectives. Ces fonctions prennent en compte divers aspects du jeu tels que le jeu lui-même, les vagues de monstres, la réserve de mana, la grille du jeu, et divers autres paramètres pour gérer les interactions du joueur avec le jeu.
- Game.h : Ce module définit les types et les fonctions liés à la gestion des tours et des projectiles dans le jeu. Il contient notamment la définition des structures Projectile, Tour, Game qui représentent respectivement un projectile, une tour et la liste des tours et des gemmes. Le module propose des fonctions pour initialiser une liste de tours, ajouter une tour dans une liste de tours, effectuer des actions avec des gemmes, calculer la distance entre deux points, créer un projectile, cibler un monstre à portée, réarranger la liste de projectiles, modifier la position des projectiles, calculer les dégâts infligés, infliger des dégâts aux monstres à l'intérieur d'une certaine portée, donner un élément ou créer une réaction élémentaire, et appeler l'action de toutes les tours. Il contient également une fonction pour gérer les dégâts sur l'ensemble des vagues. Le module inclut plusieurs autres modules tels que "gemmes.h", "vague.h", et "terrain.h" pour utiliser leurs structures et fonctions respectives.

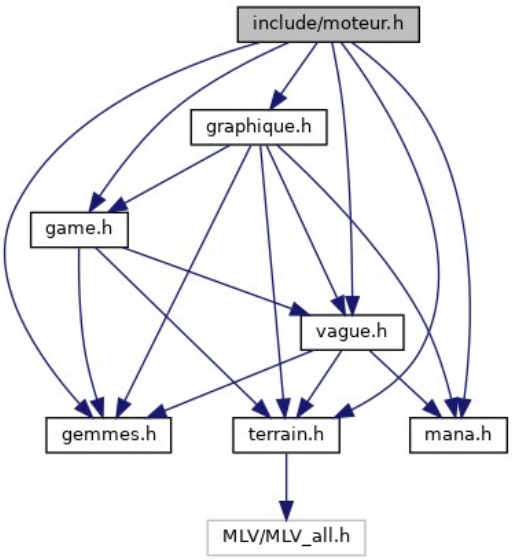

Figure 1: Graphe d'include (sans le main)

2. Choix d’implémentation

Nous avons commencé le projet par la génération du terrain et comme recommandé dans le sujet, nous avons utilisé un enum (caseType) pour représenter les cases du terrain. Ainsi nous avons fait une fonction initialise_grille qui initialise l’ensemble des cases du jeu à EMPTY. EMPTY qui représente une case vide nous sert à de nombreuses reprises comme par exemple lors du calcul de l’étendue du chemin des monstres vers une direction donnée (Nord, Sud...) ou bien lors de l’installation d’une tour représentée par TOUR. Ensuite la case aléatoire choisie au début du jeu est initialisé à MONSTER c’est l’énum qui représente le nid des monstres, par la suite chaque case constituant le chemin est initialisée à BUSY sauf la dernière case qui elle est initialisée à JOUEUR, elle représente le champ que joueur doit défendre face aux horribles monstres. L’algorithme de la génération du terrain est globalement mis en œuvre dans deux fonctions (chemin et direction), nous avons également des fonction intermédiaires que nous avons considérés comme des fonctions privées donc ils apparaissent pas dans le .h, elles ont pour but de calculer l’étendue du chemin vers une direction donnée.

Ensuite après avoir réussi la génération du terrain, nous avons choisi de faire le déplacement des monstres donc les vagues, entre-autre le fait d’initialiser les données des monstres avec la fonction init_monster ou bien encore celle de la vague avec la fonction init_vague qui appelle la fonction type_vague pour savoir quel type de vagues de monstres devrait sortir entre normale, agile, foule et boss, nous avons fait une fonction coordonnee_monstre qui parcours la grille et renvoie les coordonnées MONSTRE c’est-à-dire la case de départ des monstres(les coordonnées des monstres sont donc au début toutes à MONSTER), cette fonction permet également de trouver la case JOUEUR qui nous permettra plus tard de vérifier qu’un monstre a atteint la case à défendre ainsi le bannir à condition d’avoir suffisamment de mana. Maintenant que les monstres ont leur position de départ, nous devons leur montrer la direction à suivre car ils ont un chemin prédéfini à parcourir (chemin qu’ils doivent obligatoirement respecter), c’est dans ce sens que nous avons fait une

fonction depart_monstre qui vérifie les quatres cases autour de la case de départ et renvoie celle qui est initialisée à BUSY c'est la prochaine position des monstres. Nous faisons le déplacement des monstres à l'aide de fonction déplacement_monstre qui met à jour la position des monstres, vérifie qu'un monstre a atteint la case JOUEUR et appelle la fonction changement_direction pour vérifier à quel moment les monstres doivent changer de direction (et oui c'est pas une ligne droite en fait). Pour faire le changement de direction d'un monstre, nous vérifions d'abord qu'il se trouve au milieu d'une case pour cela nous avons utilisé la fonction fabs du module math pour vérifie qu'il est vraiment au milieu de la case. Ensuite nous avons fait quelques fonctions comme ensemble_de_vague qui permet d'initialiser toutes les vagues tout en vérifiant que les boss ne viennent pas parmi les cinq premières vagues ou bien deplacement_ensemble qui permet le déplacement de toutes les vagues de monstres créées. D'ailleurs l'ensemble de vague se trouve dans un tableau statique dans la structure TableVague, nous avons utilisé un tableau statique pour les vagues car nous pensons qu'il n'était pas nécessaire d'allouer de espace à chaque nouvelle vague ou bien d'allouer un grand espace et faire des réallocation en cas de bésion, tout cela consomme de la ressource et pas du tout nécessaire dans notre cas. Enfin nous avons fait les fonctions affiche_monstre et affiche_ensemble dans le module graphique pour pouvoir visualiser le déplacement des monstres.

Après avoir fait bouger les monstres, nous avons commencé à poser des tours avec quelques fonctions se trouvant dans le module moteur qui manipulent la bibliothèque MLV comme la fonction event_clavier qui vérifie qu'on a appuyé sur espace (cela à pour conséquence de déclencher une vague), sur q (pour quitter le jeu) et ou bien sur t (pour sélectionner une tour). Après avoir sélectionné une tour, on peut essayer de la poser en utilisant la fonction placement_tour_terrain pour vérifier que la case ou l'on veut placer la tour est vide (EMPTY), si c'est le cas cette dernière appelle la fonction AjoutTour du module game qui ajoute une nouvelle tour dans la liste des tours, les trois premières tours sont gratuites après elle vérifie que le joueur a assez de mana pour payer et la tour peut être posé. Ainsi la case EMPTY devient la case TOUR. Pour pouvoir visualiser le placement des tours, nous avons complété le module graphique avec une fonction affiche_tour qui permet l'affichage de toutes les tours posées. Après le placement des tours, nous avons commencé le module gemme par la création des gemmes en initialisant leur élément (PYRO, HYDRO, DENDRO) de façon aléatoire (une valeur entre 0 et 3) puis leur teinte dans l'intervalle donnée.
Après la création des gemmes et quelques manipulation MLV notamment un clique droit pour sélectionner une gemme, nous pouvons les posées sur les tours avec la fonction placement_gemme(elles peuvent être placé que sur les cases tours) si la tour est vide place la gemme et la liste des gemmes est réarrangée en utilisant la fonction rearrange_gemmes (déplacement des éléments d'une liste après suppression) sinon essaye une fusion avec la fonction gemme_fusion qui vérifie que le mana du joueur est supérieur à 100 pour pouvoir faire une fusion avec la fonction fusion qui fait les changements nécessaire comme la nouvelle couleur de teinte… Une fusion est possible que si les gemmes sont du même niveau. Nous avons ensuite complété le module graphique en ajoutant une fonction affiche_gemmes pour pouvoir voir la gemme avec une couleur donnée par la fonction get_color qui renvoie une couleur MLV selon la teinte de la gemme. Cette fonction est appelée dans affiche_tour si jamais une tour possède une gemme.

Ensuite, on a travaillé sur le fonctionnement des tours. On a d'abord la fonction action_tour qui va gérer les actions des tours en commençant par appeler la fonction AtRange pour chaque tour. Cette fonction va chercher à vérifier si des ennemis sont à portée puis elle choisi le monstre qui va être visé et vérifie aussi si la tour est en état de tirer avant d'appeler la fonction create_projectile qui va créer un projectile pour la tour. Ensuite la fonction déplacement_projectiles qui va gérer les actions des projectiles tirés par la tour. Elle va d'abord vérifier si la distance entre le projectile et sa cible est inférieur ou non à la taille de 3 case de la grille. Si ce n'est pas le cas, on rapproche le projectile de sa cible mais si c'est le cas, inflige des degats au monstre ciblé en fonction de la teinte de la gemme

et de son type puis vérifie si la gemme est de type pure et appel la fonction reaction_elem. La fonction reaction_elem va d'abord vérifier l'état du monstre et agir selon le résultat. Si le monstre n'est pas affecté par un état, la fonction va uniquement activer l'effet lié à l'élément de la tour. Mais si le monstre est affecté par un état, si l'état est le même que l'élément de la gemme, on va prolonger l'effet sur le monstre sinon on active la réaction élémentaire lié à la fusion des deux élément. Par contre cette fonction ne s'activera pas si le monstre est affecté par les effets vaporisation et enracinement.

3.Difficultés rencontrées

Durant le codage de notre jeu, nous avons rencontré quelques difficultés, on a eu des problèmes par rapport aux couleurs des gemmes : La fusion créait des gemmes de mauvaise couleur dû au fait que les gemmes rouges ont un teinte qui varie entre 330 et 30, ce qui veut dire qu'il est possible d'obtenir une couleur totalement différente de ce qui était prévu. Donc on a dû rajouter des lignes contre ça. On aussi des problèmes avec les projectiles des tours qui ralentissaient plus ils se rapprochaient de leur cible et n'arrivaient pas à l'atteindre mais ça a été réglé. Nous avons du à plusieurs reprises modifier les fonctions qui gèrent la circulation des monstres parce que ces dernières ne respectaient pas le chemin ou avec des vitesses imprévisibles. **Nous avons également du utiliser update_window parce que notre frame ne marchait pas bien (Ceci est un bug).** On a un dernier problème avec des projectiles qui ne disparaissent pas et donc, qui restent sur le terrain sans bouger.

4.Mode D'emploi

Touche Espace : Permet de lancer la première vague et lancer la prochaine vague en avance.
Touche t : Permet de changer entre le mode construction de tour et le mode sélection.
Touche q : Quitter la partie.

Le terrain est affiché avec des cases grises où l'on peut placer nos tours, une case rouge qui représente le nid de monstre, une case verte représentant la base que le joueur doit protéger et des cases blanche qui correspondent au chemin que vont prendre les monstres.
Sous le terrain, on a l'affichage de la quantité de mana possédé par le joueur par rapport à la quantité maximum que le joueur peut possédé avec le niveau de la réserve de mana à gauche suivi du bouton «uppgrade» permettant d'augmenter le niveau de la réserve de mana et le niveau des gemmes créée. Pour l'améliorer, il est nécessaire d'avoir au moins un quart du mana maximum de la réserve pour pouvoir faire l 'amélioration. Ensuite on a le numéro de la dernière vague lancé. Au centre on a les cases où apparaîtront les gemmes crées et à droite, on a le bouton permettant de créer une gemme.
Au début d'une partie, tu pourras créer au trois tours gratuitement puis il faudra payer 100 de mana pour la quatrième tour et ce prix ce multipliera par deux à chaque fois. Une tour, pour attaquer, à besoin d'être équipé d'une gemme. Pour créer une gemme, il faudra avoir au moins 100 multiplié par 2 puissance le niveau de la réserve de mana en mana. Tu pourras ensuite la placer dans une tour, ou la fusionner avec une gemme de même niveau dans une tour ou dans la grille de gemmes. Attention, quand tu fusionne deux gemmes pures de même couleur, c'est à dire deux gemmes de couleurs verte, rouge ou bleu et qui n'ont pas été mixés précédemment, la nouvelle gemme restera pure, mais si tu fusionnes deux gemmes de couleurs différentes, tu obtiendra une gemme mixte et cela, pour toujours.

Conclusion:
Nous avons essayé de respecter au mieux l’énoncé du sujet malgré quelques bug comme la framerate...