Rapport du Projet de Programmation C : Tower Defense

Florian Baudin, Anaëlle Fourré

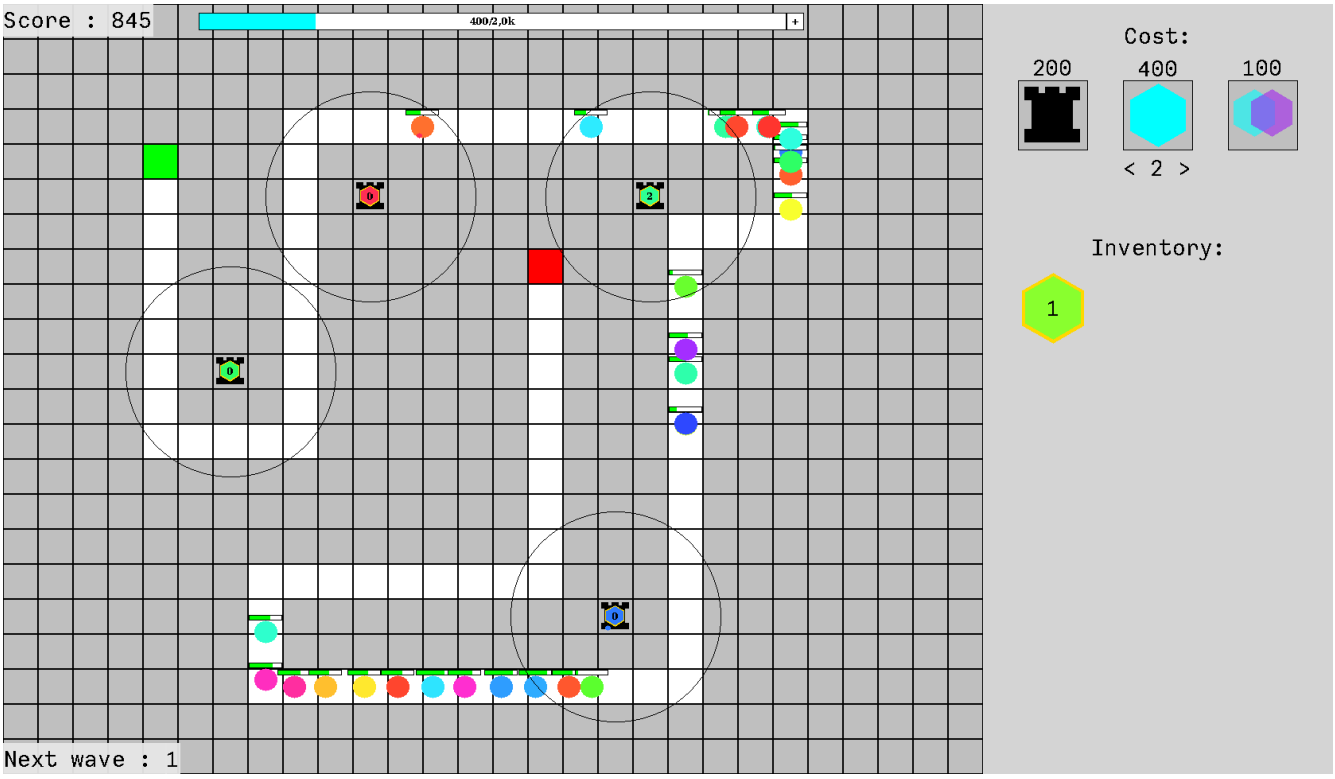

Figure 1: Copie d'écran du programme

Introduction

Objectif du projet

Lors de ce projet, nous devions réaliser un jeu de Tower Defense en C. Le but du jeu est de défendre une base contre des vagues d'ennemis en construisant des tours de défense. Le but du joueur est alors de survivre le plus longtemps possible. Afin de réaliser ce projet, nous devions suivre un cahier des charges détaillé dans le sujet, et respecter un certain nombre de contraintes. Nous devions notamment utiliser la bibliothèque graphique MLV, et respecter une architecture modulaire du projet.

Organisation du travail

Nous avons travaillé sur ce projet en binôme. Nous avons donc utilisé Git pour gérer le versionning du projet, comme indiqué dans le sujet. Nous avons en particulier travaillé sur la branche develop. Nous avons également utilisé Discord pour communiquer et nous organiser, après s'être mis d'accord sur une première structure et modularisation du projet.

Manuel utilisateur

Compilation

Afin de pouvoir jouer au jeu, il faut tout d'abord compiler le projet. Pour cela, il faut se placer à la racine du projet et exécuter la commande `make` . Cela va créer un exécutable `tower-defense` à la racine du projet ainsi que les fichiers objets intermédiaires (.o et .d) dans le dossier `bin` . En effet, chaque fichier source (.c) est compilé en un fichier objet (.o) et un fichier de dépendances (.d). Le fichier de dépendances permet de savoir quelles sont les dépendances d'un fichier source, et ainsi de recompiler automatiquement les fichiers sources qui ont été modifiés.

Exécution

Une fois le projet compilé, il suffit d'exécuter la commande `./tower-defense` pour lancer le jeu. Aucune option n'est disponible pour le moment.

Nettoyage

Afin de nettoyer le projet, il suffit d'exécuter la commande `make clean` . Cela va supprimer les fichiers objets intermédiaires (.o et .d) dans le dossier `bin` .

Commandes

Afin de jouer au jeu, il faut utiliser la souris et le clavier.

La souris permet de cliquer sur les boutons du menu, et de placer les tours sur la carte et des gemmes sur la carte. Il est alors possible de :

- Placer des tours sur les emplacements prévus à cet effet sur la carte. Pour cela, il faut cliquer sur le bouton d'achat d'une tour dans le menu, puis cliquer sur l'emplacement de la carte où l'on souhaite placer la tour.
- Acheter une nouvelle gemme pure. Pour cela, il faut cliquer sur le bouton d'achat d'une gemme dans le menu, la gemme crée sera alors placée dans l'inventaire. Le niveau de la gemme a acheté est indiqué dans le menu. Pour changer ce menu, il est possible de cliquer sur les flèches à côté du niveau de la gemme. Le niveau maximal d'une gemme pure achetée est de 32.
- Placer une gemme sur une tour. Pour cela, il faut cliquer sur la gemme dans l'inventaire, puis cliquer sur la tour sur laquelle on souhaite placer la gemme. Si la tour possède déjà une gemme, celle-ci sera remplacée par la nouvelle gemme. Un temps de recharge de 2 secondes s'applique une fois la gemme ajoutée dans le tour.
- Enlever une gemme d'une tour. Pour cela, il suffit de cliquer sur la tour dont on souhaite retirer la gemme. La gemme est alors placée dans l'inventaire.
- Fusionnner deux gemmes. Pour cela, il faut d'abord cliquer sur le bouton de fusion de gemmes, puis sur la première gemme dans l'inventaire, et enfin cliquer sur la seconde gemme dans l'inventaire. Si les deux gemmes sont fusionnables, alors une nouvelle gemme est créée et placée dans l'inventaire.
- Augmenter le niveau de la barre de mana. Pour cela, il suffit de cliquer sur le bouton "+" à côté de la barre de mana. Cela coûte un quart du niveau maximal de mana.

Le clavier permet de :

- Lancer la première vague d'ennemis. Pour cela, il suffit d'appuyer sur la touche "Espace" du clavier. Une fois la première vague lancée, une vague sera lancée toutes les 35 secondes. Il est cependant possible de lancer une vague manuellement en appuyant sur la touche "Espace" (à partir de 25 secondes restantes avant la prochaine vague), ce qui fera gagner du mana.
- Quitter le jeu. Pour cela, il suffit d'appuyer sur la touche "Q" du clavier.

Si un ennemi atteint la base, et qu'il n'y a pas assez de mana pour bannir le monstre, la partie se termine et un message s'affiche pour indiquer que le joueur a perdu. Il suffit alors de cliquer sur n'importe quel bouton pour quitter le jeu.

Modularisation du projet

Le projet est organisé en plusieurs modules, chacun ayant un rôle bien défini.
Voici la liste des modules :

- Les modules contenant les structures principales des éléments du jeu et leur gestion:
  - `Map` : contient la structure de la carte du jeu et les fonctions permettant de gérer la carte
  - `Monsters` : contient la structure des monstres du jeu et les fonctions permettant de gérer les monstres
  - `Shots` : contient la structure des projectiles du jeu et les fonctions permettant de gérer les projectiles
  - `Element` : contient la structure des teintes et éléments du jeu et les fonctions permettant de gérer les éléments
  - `Gems` : contient la structure des gemmes du jeu et les fonctions permettant de gérer les gemmes
  - `Mana` : contient la structure de la barre de mana du jeu et les fonctions permettant de gérer la barre de mana
- Les modules permettant la gestion du jeu:

  - `Game` : contient la structure du jeu et les fonctions permettant de gérer le jeu
  - `Wave` : contient les fonctions permettant de générer les vagues d'ennemis
  - `Error` : contient la structure et les fonctions permettant de gérer les erreurs du jeu
  - `Inventory` : contient la structure de l'inventaire du jeu et les fonctions permettant de gérer l'inventaire
  - `Timer` : contient la structure du timer du jeu et les fonctions permettant de gérer le framerate du jeu
  - `TowerDefense` : contient la boucle principale de jeu
- Les modules permettant l'affichage du jeu:
  - `Window` : contient la structure de la fenêtre graphique du jeu et les fonctions permettant de gérer la fenêtre graphique et ses informations
  - `Display_Buttons` : contient les fonctions permettant d'afficher les boutons du jeu
  - `Display_Info` : contient les fonctions permettant d'afficher les informations du jeu
  - `Display_Map` : contient les fonctions permettant d'afficher les éléments et la carte du jeu
  - `Color` : contient les fonctions permettant de gérer les couleurs pour l'affichage du jeu
  - `Graphic_Utils` : contient les fonctions permettant d'afficher des formes utilisées dans l'affichage du jeu
  - `Graphic` : contient les fonctions permettant d'afficher la totalité du jeu
- Les modules permettant la gestion des événements de l'utilisateur:
  - `Event` : contient les fonctions permettant de gérer les événements de l'utilisateur dans la fenêtre graphique
  - `Action` : contient les fonctions permettant de gérer les actions de l'utilisateur dans le jeu
- Les modules utilitaires :
  - `Utils` : contient les structures utilitaires du jeu et les fonctions permettant de gérer ces structures, tel que les coordonnées, les directions et les vecteurs.
  - `Queue` : contient les macros permettant la création et la gestion de liste chainée du jeu

Choix d'implémentation

Structures de listes chainées

Dans notre structure de jeu, nous avons décidé de stocker les monstres, les tirs et les gemmes actives dans des listes chainées. Ainsi, nous pouvons facilement ajouter et supprimer des éléments de ces structures, et nous pouvons facilement parcourir ces structures pour afficher les éléments, ou pour gérer les collisions. Nous avons utilisé les macros du module `sys/queue.h` pour gérer ces listes chainées, afin de pouvoir facilement créer des listes chainées pour ces différents structures. `sys/queue.h` n'étant pas totalement standard, nous avons simplement copié les macros dans notre module `Queue` .

Gestion du chemin des monstres

La génération aléatoire du chemin des monstres est faite avec l'algorithme indiqué dans le sujet. Le chemin est ensuite stocké dans le module `Map` , en changeant le type des cellules de la carte faisant partie du chemin et en y ajoutant leur direction. Nous stockons également dans la map les coordonnées du nid des montres. Afin de parcourir le chemin, il nous suffit donc de partir du nid des monstres et de suivre le chemin grâce aux directions de chaque cellule. De même, nous pouvons facilement calculer la prochain cellule où le monstre doit se déplacer.

Tours et gemmes

Lorsque l'on ajoute une tour à la map, on modifie le type de la cellule de la carte du module `Map` . On y ajoute également la gemme si une est positioné dessus.

Lorsqu'une gemme est positionnée sur une tour, on ajoute la gemme comme "gemme active" dans une liste chainée de gemmes actives. Ces à partir de cette liste qu'on créer les tirs en les parcourant.

Mouvemement des monstres et des tirs

Pour gérer le mouvement des monstres et des tirs, nous avons utilisé le temps calculé dans la boucle principale du jeu, correspondant au temps de dernière mise à jour. Cela permet de savoir combien de temps s'est écoulé depuis

la dernière mise à jour et donc de déterminer la distance parcourue par le monstre ou le tir depuis la dernière mise à jour.

Stockage des tirs

Les tirs sont stockés dans une liste chainée sur les monstres ciblés. Ainsi, si le monstre meurt en étant touché par un de ces tirs, tous les autres tirs le ciblant peuvent être supprimés facilement.

Gestion des vagues

Afin de gérer les vagues, notre module `Wave` nous permet la génération des vagues d'ennemis, en fonction du niveau de la vague. Nous avons décidé de ne pas avoir de structures de vagues et de stocker les monstres de toutes les vagues dans une seule liste chainée. Nous avons en effet observé que les informations de la vague dont le monstre provenait n'etait utile que lors de la création du monstre, et que ces informations n'étaient plus utiles par la suite, les monstres étant ensuite gérés de manière indépendante.

Gestion de la bibliothèque graphique MLV

Afin d'afficher le jeu et de lire les événements, nous avons utilisé la bibliothèque graphique MLV. Nous avons pris le soin de définir des modules précis qui gèrent l'affichage et les événements de la bibliothèque graphique, et qui sont les seuls à inclure la bibliothèque graphique. Ainsi, les autres modules n'ont pas besoin de connaitre la bibliothèque graphique, et peuvent être utilisés sans la bibliothèque graphique.

Frame rate et timer

Pour la gestion du temps dans le jeu, nous avons utilisé les structures de temps et les fonctions de `time.h` . Ainsi, nous initialisons un timer au début du jeu, et nous calculons pour les structures le nécessitant, le temps future auquel l'action se fera. Nous pouvons ainsi déterminer si une nouvelle vague doit être lancée, si un monstre doit se déplacer, ou si une gemme a terminé de charger, chacune des ces structures contenant un temps auquel il doivent se lancer, il suffit donc de regarder si ce temps a été dépassé.

Gestion des événements

Afin de gérer les événements d'entrées utilisateur, nous avons utilisé la bibliothèque graphique MLV. Nous récupérons ainsi les clics de la souris et clavier dans notre boucle principale, et ils sont ensuite traités par le module `Action` qui gère l'événement en fonction de l'action courante du joueur et qui détermine ensuite la nouvelle action du joueur.

Gestion des erreurs

Afin de gérer les erreurs de l'utilisateur, nous avons créé une structure `Error` . Cette structure contient un type d'erreur, et un temps de fin. Ainsi, lorsque l'utilisateur fait une erreur, nous créons une erreur avec le type d'erreur correspondant, et le temps de fin correspondant à un temps de 3 secondes dans le futur. Nous pouvons ainsi afficher l'erreur à l'écran à chaque affichage, et la faire disparaitre lorsque le temps de fin est dépassé.

Conclusion

Nous avons créé un jeu de Tower Defense en C, en essayant de maintenir un code propre et bien organisé. Le programme final est jouable, bien qu'un peu ennuyant car assez facile une fois le jeu compris. En effet, beaucoup de formule dépendent du niveau de la réserve de mana. Ainsi, si l'on démarre une vague avec beaucoup d'avance on gagne assez de mana pour l'agrandir et donc en gagner encore plus en en démarrant encore une avec beaucoup d'avance. Le mana est suffisant pour acheter des gemmes très fortes et battres tous les monstres de plusieurs vagues simultanément. Nous avons essayé de contrer cette facilité en ajoutant un délai de dix secondes entre le démarrage de deux vagues et en diminuant la constante de dégât laissé à notre choix. Le développement de ce jeu n'en était pas moins intéressant.