Tower Defense

MAUREL Victorine - SEGUY Margaux

Introduction

Ce projet alliant programmation algorithmique et synthèse de l'image a pour objectif de nous faire coder un jeu : un Tower Defense. Nous avons un cahier des charges complet et précis mais surtout chargé de fonctionnalités. Notre binôme se constitue de Victorine Maurel et Margaux Seguy. Nous avons commencé à travailler sur ce projet peu de temps après avoir reçu les consignes. La répartition des tâches s'est faite assez naturellement, nous travaillons donc chacune sur des fonctions différentes. Nous avons mis en place un répertoires GitHub pour synchroniser facilement notre travail et voir l'avancée du projet dans sa globalité. Ce projet, bien que très intéressant, nous a demandé énormément d'heures de travail ; malgré un délai de rendu supplémentaire nous regrettons de ne pas pouvoir le finir ; non pas par manque de temps ou d'implication mais plutôt par manque de compétences.

Notre principale difficulté a été de comprendre ce qui nous était demandé pour réaliser ce projet. En effet, nous avons rapidement compris l'objectif du programme mais la mise en place des différentes étapes pour son exécution a été plus longue et compliquée. Nous avons eu du mal à comprendre la relation entre les différentes fonctions que nous devions programmer. Mais surtout nous ne savions pas par où commencer. Le programme semble abstrait tant que nous ne pouvons pas le tester, et pour le tester il faut avoir réalisé un certain nombre de fonctions – les premières heures de travail ont donc été déstabilisantes pour nous. Pour faciliter notre travail et permettre une meilleure compréhension du projet nous avons décidé de mettre en place un schéma montrant explicitement la relation entre les différentes fonctions, schéma que nous complétions au fur et à mesure de l'avancée du projet. Voir **annexe 1**.

Malgré une bonne organisation, une répartition des tâches judicieuse et une forte implication dans ce projet nous n'avons pas réussi à mettre toutes nos idées en oeuvre et par conséquent nous ne pouvons pas rendre une application complètement aboutie. Fort de cette considération, nous allons nous appliquer dans la rédaction de ce rapport pour vous présenter notre projet et ses résultats.

Sommaire

Présentation de l’application

Les Tower Defense en général

Le but de ce genre de jeu est de défendre une zone contre des vagues d’ennemis. Les vagues sont composées de monstres. Toutes les vagues ne sont pas forcément les mêmes, elles peuvent comporter différents types de monstres.
Il est possible au fil de la partie d’acheter des tours et également d’améliorer celles déjà présentes à l’aide de bâtiments.
Chaque ennemi éliminé rapporte des points au joueur, qu’il va pouvoir utiliser pour acheter de nouvelles tours. Ces points peuvent être assimilés à de l’argent virtuel.
Ces dernières ont des caractéristiques telles que leur coût, leur vitesse, leur portée ou encore leur type. Les montres sont quant à eux caractérisés par leur rapidité de déplacement ou encore leur immunité face à certains types d’attaques.

Notre Tower Defense

Notre application de Tower Defense est basée sur le concept même du jeu. Avec notamment une carte, des tours, des monstres et des bâtiments. Pour ce qui est du design nous avons préféré le laisser de côté et nous concentrer en priorité sur les parties algorithmique et infographie. Ce qui explique l’aspect plutôt simpliste de nos différents éléments en effet nous avons préféré nous contenter de design assez basiques et commun pour des Tower Defense (voir **annexe 2).** Pour ce qui est des éléments nous nous sommes contentées de simple forme géométrique.

Le jeu se lance à partir du terminal avec la commande : *bin/towerdefense*. Une fenêtre s'ouvre alors et un menu apparaît qui propose de lancer le jeu ou d'accéder à l'aide. L'interface du jeu en lui même se compose principalement de la carte. L'appui sur la touche h permet à l'utilisateur d'avoir accès à l'aide, et savoir ce qu'il doit faire pour entre autre lancer la partie, placer une tour, augmenter la force de ces dernières ou encore en acheter. Ces règles seront détaillées dans la partie fonctionnalités (**Section 5**).

Nous avons pris le parti d'utiliser les touches du clavier plutôt que des boutons pour un souci de simplicité. Nous avons conscience qu'il aurait été préférable pour l'expérience utilisateur d'utiliser des boutons afin qu'il sache directement ce qu'il doit faire pour provoquer telle action ou fonctionnalité.
De plus l'application en elle même, et donc le jeu, n'est pas réellement fonctionnelle. C'est à dire que toutes les méthodes que nous voulions créer pour faire fonctionner le jeu ont été implémentées, et sont donc présentes dans nos différents fichiers (voir **annexe 3)**, mais elles ne sont pas toutes utilisables par le joueur.
Pour l'instant il peut lancer la partie, et donc créer une vague de monstre, et placer des tours et des bâtiments.

Description globale de l’architecture

Au niveau de l’architecture de l’application nous avons respecté la demande du cahier des charges. Nous avons donc créé chaque entité avec des fichiers d’en-tête .h et des fichiers .cpp contenant le code à proprement parlé. Nous avons également regroupé tous les fichiers d’en-tête dans un dossier “include” et tous les fichiers de code dans un dossier “src”. Toutes les images sont placées dans un dossier “image” et l'exécutable final est placé dans un dossier “bin”. Comme demandé le fichier itd est rangé dans un dossier nommé “data” et tous les documents jugés utiles ainsi que ce rapport sont placés dans un dossier “doc”. Un Makefile a également été créé pour faciliter la compilation de l’application.

Nous avons découpé notre code en plusieurs entités pour faciliter sa compréhension, limiter la taille du fichier et améliorer la durée de traitement. Pour cela nous nous sommes basées sur un design pattern de forte cohésion. Dans le tableau ci-dessous vous pouvez voir toutes les entités que nous avons utilisées. Nous avons choisi de découper ces entités de manière assez logique, c’est à dire qu’elles contiennent chacune un élément qui lui est propre. Certaines de ces entités contiennent des includes vers d’autres entités de manière à pouvoir réutiliser des paramètres d’autres fonctions. Voir **annexe 1**.

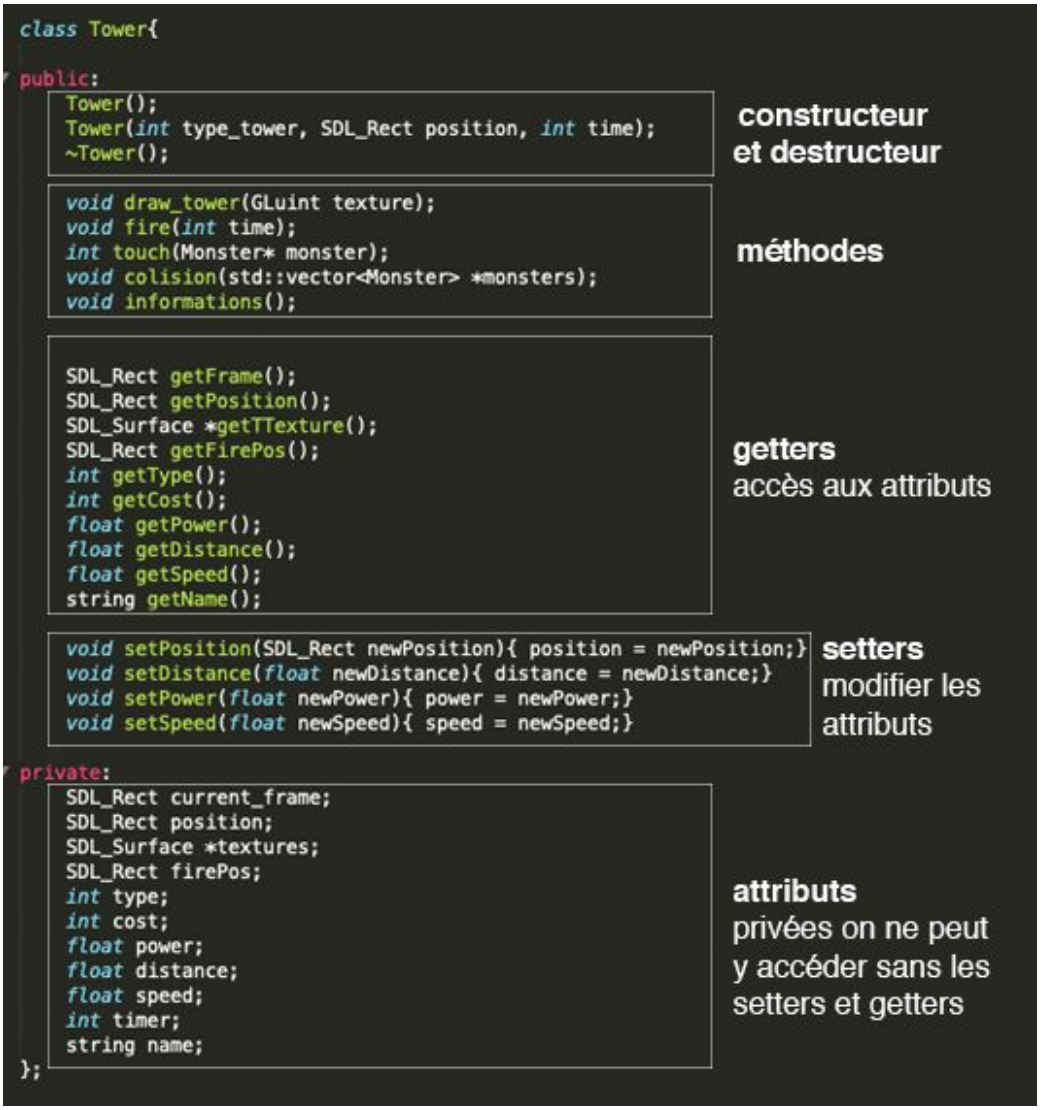
building.h	building.cpp	Pour la gestion des bâtiments
checkMap.h	checkMap.cpp	Vérification de la carte et de l’itd
include.h		Contient toutes les librairies et variables nécessaires au projet
	main.cpp	Fichier général
map.h	map.cpp	Affichage de la carte
monster.h	monster.cpp	Pour la gestion des monstres
timer.h	timer.cpp	Pour la gestion du temps et des pauses
tower.h	tower.cpp	Pour la gestion des tours
utils.h	utils.cpp	Pour la gestion des textures
wave.h	wave.cpp	Pour gérer le lancement du jeu et le nombre de vagues

Description des structures de données

Pour implémenter notre Tower Defense nous avons utilisé le langage C++, qui nous a permis d'utiliser des classes. Cela a rendu plus facile l'accès aux différents éléments dont nous avions besoin mais également la compréhension du code pour une personne lambda qui le lit. Un autre avantage de l'utilisation de classes est que la fragmentation du code est plus simple et semble même quasiment évidente en associant à une entité les méthodes et attributs qui lui correspondent.

Chaque entité nécessaire à la réalisation du Tower Defense possède une classe, avec toutes les méthodes et les attributs qui lui appartiennent. Comme il est souvent le cas en programmation orientée objets tous nos attributs sont privés, afin de les protéger. Pour pouvoir y accéder ou les modifier nous avons mis en place des getters et des setters. Chaque instance de nos classes est créée à l'aide d'un constructeur et est détruit avec un destructeur, qui sont définis avec la classe dans les différents .h .

L'exemple de tower.h (toutes nos classes sont définies de la même manière) :



Description des fonctionnalités du jeu

Cette application requiert beaucoup de fonctions pour être entièrement fonctionnelle, interactive et proposer une bonne expérience utilisateur. Nous nous sommes donc appliquées à créer toutes les fonctions que nous jugions nécessaires pour chacune des entités que nous avons présentée à la **section 3**. Les entités contiennent donc des classes avec des constructeurs et destructeurs ainsi que leurs fonctions propres associées.

Une fois les différentes fonctions implémentées il faut les appeler et les mettre en relation avec d'autres entités de manière à les rendre fonctionnelle et pouvoir les afficher à l'aide de OpenGL. Bien que nous ayons implémenté un certain nombre de fonctions pour les différentes entités, nous ne les avons pas toutes utilisées et leurs effets ne se voient pas au lancement du jeu. En **annexe 3** vous pouvez trouver toutes les fonctions que nous avons implémentées.

Concernant les règles du jeu, nous avons repris les règles d'origine qui sont expliquées dans la **section 2**. Nous avons mis en place des touches de commandes pour diriger le jeu. Tout d'abord, pour lancer le jeu l'utilisateur peut soit cliquer sur le bouton *jouer* visible à l'écran soit appuyer la barre espace de son clavier. Il peut également obtenir les informations essentielles telles que les touches de commandes grâce à l'*aide*. Pour cela il peut soit cliquer sur le bouton *aide* visible à l'écran qui affichera l'aide directement dans sa fenêtre de jeu soit appuyer la touche *h* qui affichera l'aide dans le terminal, de manière à ce qu'il puisse jouer dans la fenêtre principale tout en ayant l'aide visible. L'utilisateur peut également mettre le jeu en pause en appuyant sur la touche *p* et il lui suffit de réappuyer sur cette même touche *p* pour reprendre le jeu. Pour afficher les informations supplémentaires, dans le terminal, telles que le temps, le nombre de vagues ou l'argent disponible il faut appuyer sur la touche *i*. Pour afficher une tour rocket l'utilisateur doit appuyer sur la touche *a*, pour une tour laser sur la touche *z*, pour une tour jaune sur la touche *e*, pour une tour bleue sur la touche *r*. Pour afficher un bâtiment radar il doit appuyer sur la touche *t*, pour un bâtiment usine d'armement sur la touche *y*, pour un bâtiment stock de munitions sur la touche *u*. Le fait de cliquer sur une touche pour créer des bâtiments ou des tours contraint l'utilisateur sur la position de ces dernières. Le fait que l'utilisateur puisse choisir où poser les éléments est une des améliorations à envisager.

Nos éléments

Les Tours :

Nom	Coût	Pouvoir	Portée	Vitesse
Rouge	100	80	50	20
Verte	100	50	20	80
Jaune	30	50	10	60
Bleue	50	30	65	65

Les Bâtiments :

N°	Coût	Portée	Attribut tour modifié
1	150	50	Portée (+25%)
2	120	50	Pouvoir (+25%)
3	100	50	Vitesse (+25%)

Les Monstres :

N°	Vitesse	Résistance	Gain	Vie
1	1	2	5	80
2	2	1	7	60

Conclusion

Nous avons donc une application capable d'afficher un menu d'accueil avec des boutons, une page d'aide, une carte de jeu, des tours et des bâtiments ainsi que l'affichage des monstres. Nous pouvons interagir entre ces différents éléments à l'aide des touches du clavier. Notre jeu n'est malheureusement pas terminé, les principales fonctionnalités manquantes sont : l'algorithme du plus court chemin, l'utilisation de l'algorithme de Bresenham pour vérifier la validité des noeuds ainsi que la gestion des tirs et collisions.

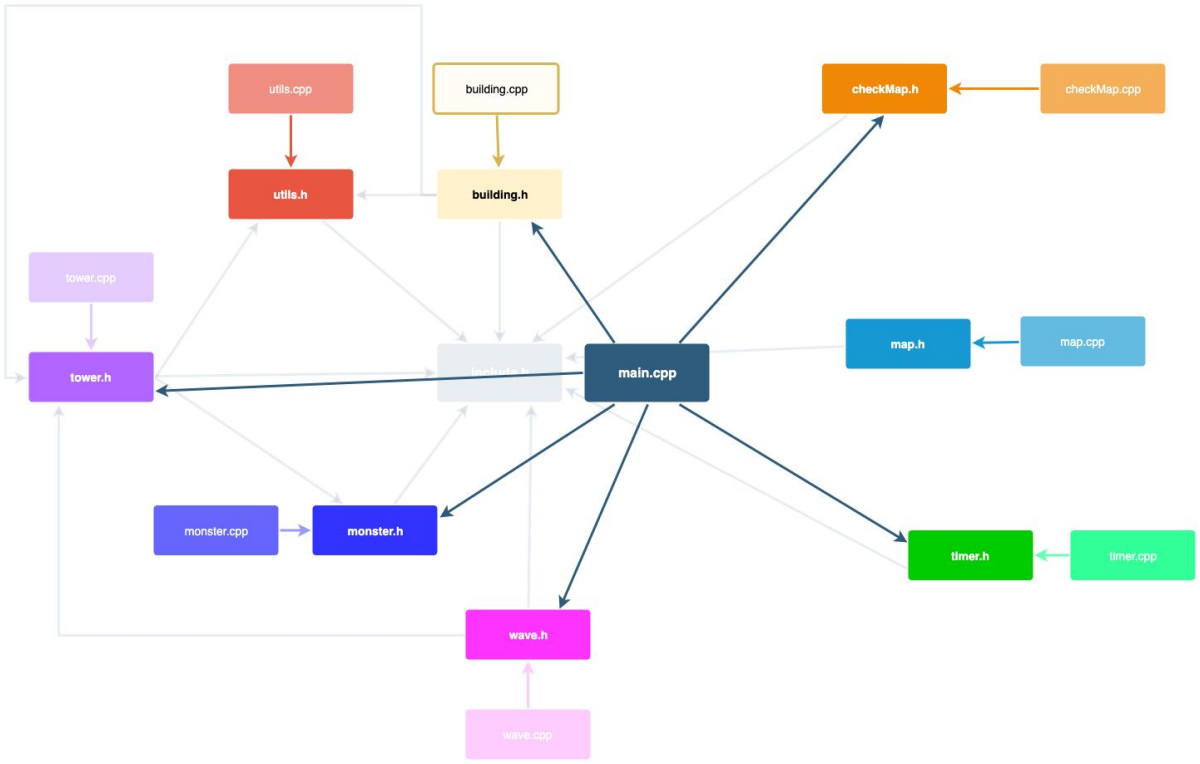
Notre jeu pourrait également subir un certain nombres d'améliorations telles que :

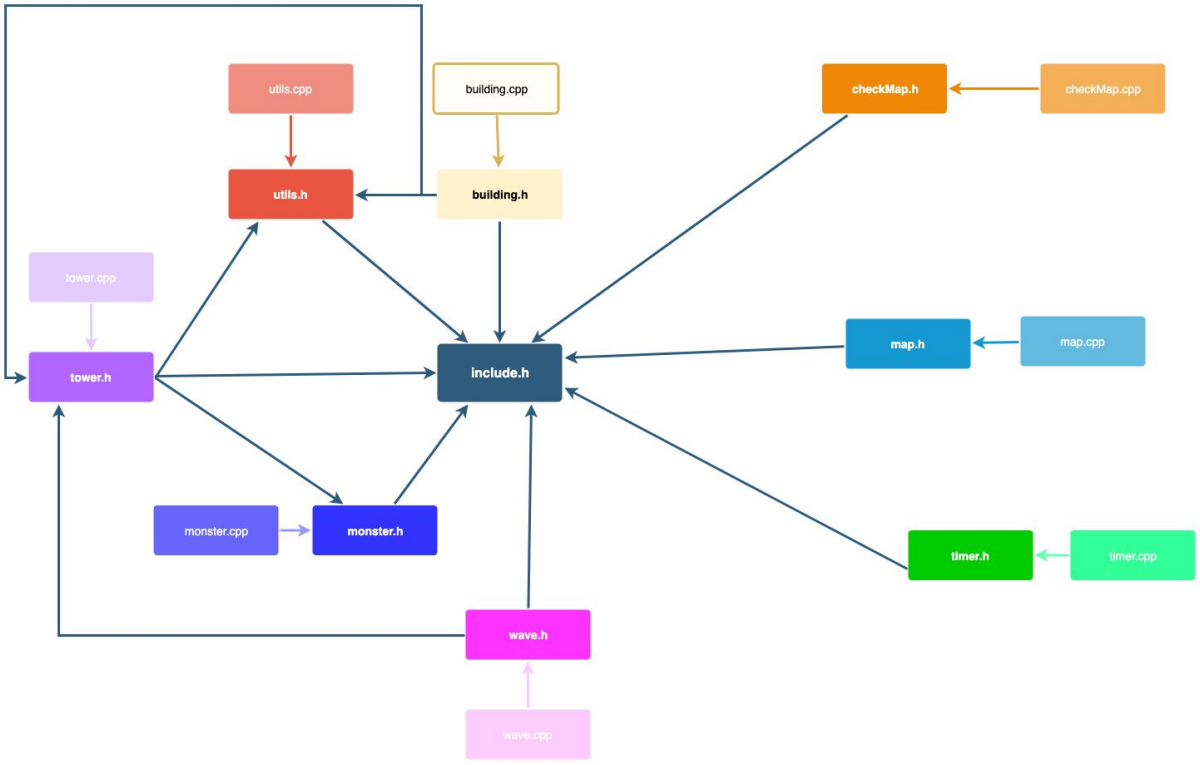
- une meilleure gestion du temps (la pause ne met pas réellement le jeu en pause)
- une meilleure gestion de l'argent
- une meilleure gestion des vagues de monstres et des niveaux
- choisir la position des tours et bâtiments
- améliorer le graphisme
- améliorer l'expérience utilisateur (avec du son par exemple)
- une gestion du game play dans un fichier fait pour cela et du coup un main plus propre et organisé
- gestion du texte dans la fenêtre pour afficher le temps, l'argent et le nombre de vagues

Malgré tout cela nous avons apprécié réaliser ce projet et nous avons pu améliorer nos compétences tant en C++ qu'en OpenGL, en effet avant le début de ce projet nous étions des novices en C++ il nous a permis de découvrir de très nombreuses particularités de ce langage.

Nous aurions évidemment préféré pouvoir livrer un projet totalement abouti mais nous sommes tout de même fières de ce que nous vous proposons, car nous ne pensions pas en début de projet rendre ce que nous vous rendons aujourd'hui.

Annexe 1


utils.cpp
building.cpp
checkMap.h
checkMap.cpp
utils.h
building.h
tower.cpp
tower.h
main.cpp
map.h
map.cpp
monster.cpp
monster.h
timer.h
timer.cpp
wave.h
wave.cpp


utils.cpp
building.cpp
checkMap.h
checkMap.cpp
utils.h
building.h
tower.cpp
tower.h
include.h
map.h
map.cpp
monster.cpp
monster.h
timer.h
timer.cpp
wave.h
wave.cpp

Annexe 2

Monstres :

Tours :

Bâtiments :

Annexe 3

Fichier	Fonction	Utilisation
building	Building();	oui
	void draw_building	oui
	float installation	non
	void colision	non
	int cost_building	non
checkMap	checkMap();	oui
	int checkItd	oui
	int checkMap	oui
	unsigned int getpixel	oui
	int loadPPM	oui
	void bressenham_x_y	non
map	void drawRepere	non
	void drawMap	oui
monster	Monster();	oui
	void create_monster	oui
	void draw_monster	oui
	void deplacement	oui
	void update_monster	non
	void dead_monster	non

timer *pour l'instant le temps est géré dans le main*	Timer();	non
	void start	non
	void pause	non
	void unpause	non
	void stop	non
tower	Tower();	oui
	void draw_tower	oui
	int touch	non
	void colision	non
	void informations	oui
utils	void drawCircle	oui
	void loadImage	oui
	void loadTexture	oui
wave	Wave();	oui
	void create_wave	oui