LESBATS Lucie RATHONIE Line

SAVE YUBABA'S BATHOUSE !

IMAC
INGÉNIEUR
ESIPE

Projet : Synthèse d'image IMAC 1 Année 2018-2019

SOMMAIRE

INTRODUCTION	p. 3
1. PRÉSENTATION DE L'APPLICATION	p. 3
2. FONCTIONNALITÉS	p. 4
3. ARCHITECTURE DE L'APPLICATION	p. 5
4. STRUCTURE DES DONNÉES	p. 6 et 7
5. DÉROULEMENT DU PROGRAMME	p. 7
6. RÉSULTATS OBTENUS	p. 8 à 10
7. RÉPARTITION DES RÔLES	p. 11
8. DIFFICULTÉS ET PISTES POUR CONTINUER	p. 12 et 13
CONCLUSION	p. 13

INTRODUCTION

Dans le cadre des cours Synthèse d'image et Programmation et Algorithmique, il nous a été demandé d'implémenter un Tower Defense.

L'IMAC Tower Defense est une version basique d'un tower defense. La programmation est faite en langage C++ et est commentée. Les paragraphes qui suivent rendent compte de nos choix ainsi que des différents problèmes rencontrés.

Nous avons choisi comme graphismes l'univers Ghibli car c'est un univers qui nous plaît à toutes les deux et qui nous inspirait pour le design.

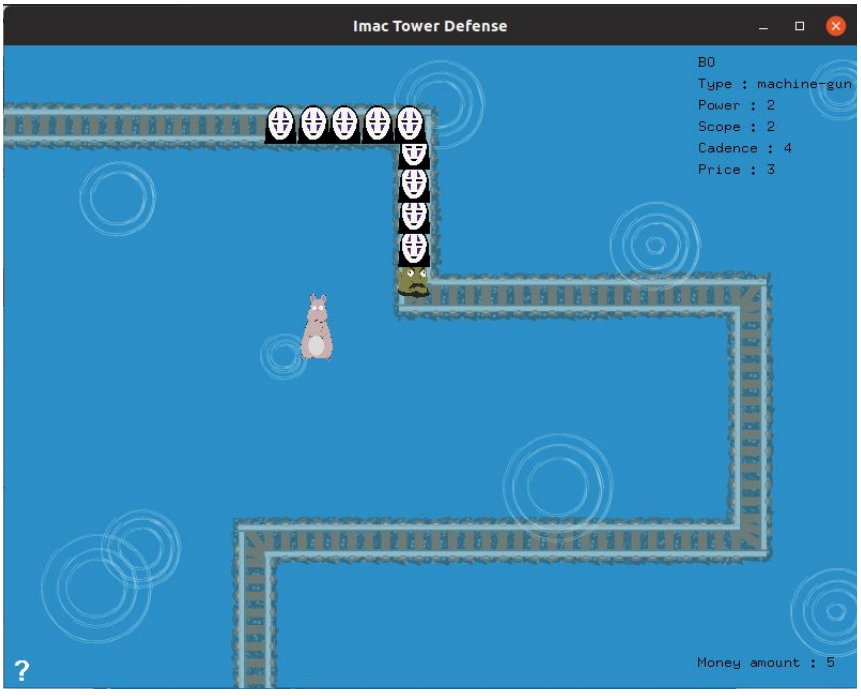
Le lien du Github de notre projet est le suivant : https://github.com/Lucie98765/imac_tower_defense

1. PRÉSENTATION DE L'APPLICATION

Notre application est un jeu de Tower Defense inspiré du film d'animation Le voyage de Chihiro, de Hayao Miyazaki.

Le Tower Defense est un jeu vidéo dont l'objectif est d'empêcher des vagues de monstres d'arriver à une zone précise. Les monstres se déplacent en suivant un ou plusieurs chemins et, pour s'en défendre, l'utilisateur peut acheter des tours et les placer sur la carte. Il peut également acheter des installations, qui amélioreront les capacités des tours (puissance, cadence, portée dans notre cas).

Nous avons choisi de coder notre application en C++ pour utiliser la programmation orientée objet et pour pouvoir créer différentes classes, pour ainsi gérer nos entités plus simplement.



2. FONCTIONNALITÉS

A implémenter	Implémentées	Fonctionnelles
Gestion de la carte		
Chargement de la carte	Oui	Oui
Vérification du .itd	Oui	Oui
Chemin le plus court	/	/
Validation de la carte		
Validation des noeuds	Oui	Oui
Validation du chemin	Oui	Partielle
Gestion des entités		
- Gestion des Tours		
Code	Oui	Oui
Représentation	Oui	Oui
Animation	/	/
- Gestion des Monstres		
Code	Oui	Oui
Représentation	Oui	Oui
Animation	Partielle	Oui
- Gestion des bâtiments		
Code	Oui	/
Représentation	Oui	Oui
Animation	/	/
Gestion du gameplay		
- Infographie		
Sprites / Textures	Oui	Oui
Interface d'accueil	Oui	Oui
Bouton et panneau d'aide	Oui	Oui
- Jeu		
Vagues	partielle	oui
Gestion de l'argent	/	/

3. ARCHITECTURE DE L'APPLICATION

Pour ce qui est de l'architecture de nos dossiers, ils sont organisés comme suit :

-bin
    ---ImacTowerDefense //exécutable
-data
    ---carte.itd
    ---carte.ppm
-doc
    ---Sujet du projet (pdf)
    ---Rapport (pdf)
-images
    ---images des monstres (png)
    ---images des tours (png)
    ---images des installations (png)
    ---image de fond/background (png)
    ---image du bouton d'aide
-include
    ---fichiers.h
-Makefile
-obj
    ---fichiers.o
-src
    ---fichier.cpp

Pour ce qui est de notre application en général, elle se découpe plus ou moins en deux grosses parties : une partie algorithmique et une partie de gestion de l'affichage avec SDL.

La partie algorithmique de notre projet gère tout ce qui est de la logique de jeu, des calculs et des interactions entre les différentes structures. Elle gère aussi la lecture et la vérification des cartes proposées, le placement des objets dessus et les différentes fonctionnalités inhérentes aux objets.

La partie d'affichage, réalisée grâce à la SDL gère l'affichage de textures et d'images, ainsi que de textes informatifs. Elle est uniquement implémentée dans les fichiers main.cpp, texture.cpp et display.cpp alors que la partie algorithmique se retrouve dans tous les autres fichiers ainsi que dans le main.

4. STRUCTURES DES DONNÉES

A. Partie algorithmique

- **Classe Monster** : Classe permettant de créer les monstres, de gérer leurs actions et de les faire interagir avec d'autres objets. Les monstres ont diverses méthodes : ils ont évidemment des accesseurs (get et set), mais aussi une méthode leur permettant de se déplacer.
- **Classe Tower** : Classe permettant de créer les tours, de gérer leurs actions et de les faire interagir avec d'autres objets. Les tours ont diverses méthodes : elles ont évidemment des accesseurs (get et set), mais aussi une méthode leur permettant de de tirer.
- **Classe Installation** : Classe permettant de créer les installations, de gérer leurs actions et de les faire interagir avec d'autres objets. Les installations ont diverses méthodes : elles ont évidemment des accesseurs (get et set), mais aussi deux méthodes leur permettant de de gérer l'amélioration que chaque type d'installation procure à la tour à côté de laquelle elle est déposée.
- **Classe Entity** : Les trois classes présentées ci dessus sont des sous-classes de la classe Entity, une classe comprenant les attributs de positions x et y ainsi que leur getters et setters. Entity nous a aussi servi à stocker des pixels.
- **Classe Map** : Map regroupe à la fois les informations de la carte .ppm et du .itd . C'est la classe qui permet de faire vivre le jeu. A partir des map.h et map.cpp nous avons implémenté la vérification de la carte et le stockage du chemin.
- **Classe Piece :** Classe permettant la gestion de l'argent et tout ce qui concerne les achats (tours et bâtiments) ainsi que le gain d'argent (destruction des monstres).
- **Classe Node** : Les noeuds correspondent aux entrées, sorties et intersections de la carte. Cette classe contient les positions x et y des noeuds ainsi que des attributs comme leur couleur en RGB. Il ont évidemment des accesseurs.
- **Classe Timer** : La classe Timer devait nous servir pour la gestion du temps, notamment en l'associant à la vitesse des monstres et la cadence des tours, ou encore pour la gestion d'un bouton pause. Cependant, nous n'avons pas eu le temps de l'utiliser même s'il est implémenté.

B. Partie affichage avec SDL

Nous avons utilisé diverses librairies SDL pour réussir à afficher les textures, les formes et certains textes. Nous avons utilisé la structure donnée par Olivier Falconnet pour les affichages, c'est à dire une boucle de dessin, une boucle de gestion des événements et diverses fonctions de dessin. La partie d'affichage est gérée en grande partie par le fichier main.cpp, qui appelle pour ces fonctions d'affichages les fichiers display.h et texture.h, ainsi que les librairies dont nous avons eu besoin.

C. Autres

- Fichier headers.h : fichier regroupant toutes les librairies nécessaires au code. Cela nous permet d'avoir des en-têtes moins longs dans les autres fichiers.
- Makefile : fichier facilitant la compilation de notre code. Nous avons repris un Makefile déjà créé par nos professeurs et l'avons modifié et mis à jour pour qu'il convienne à notre projet.

5. DÉROULEMENT DU PROGRAMME

Pour exécuter le programme, il faut ouvrir un terminal à la racine du projet. Il faut ensuite lancer la commande " make ", puis la commande " ./bin/ImacTowerDefense ".

Lorsque l'utilisateur lance l'exécutable, il y a plusieurs étapes avant la partie "jeu":

- Lecture et stockage du .itd : Tout d'abord, on vérifie que le .itd sur lequel sera basé tout le reste du jeu est conforme aux critères.
- Lecture et stockage de la carte : Ensuite, on vérifie la carte correspondante au .itd. Pour cela on vérifie si les couleurs associées aux différents pixels de l'itd (noeuds, chemins…) sont bien les mêmes dans la carte.

La partie commence ensuite :

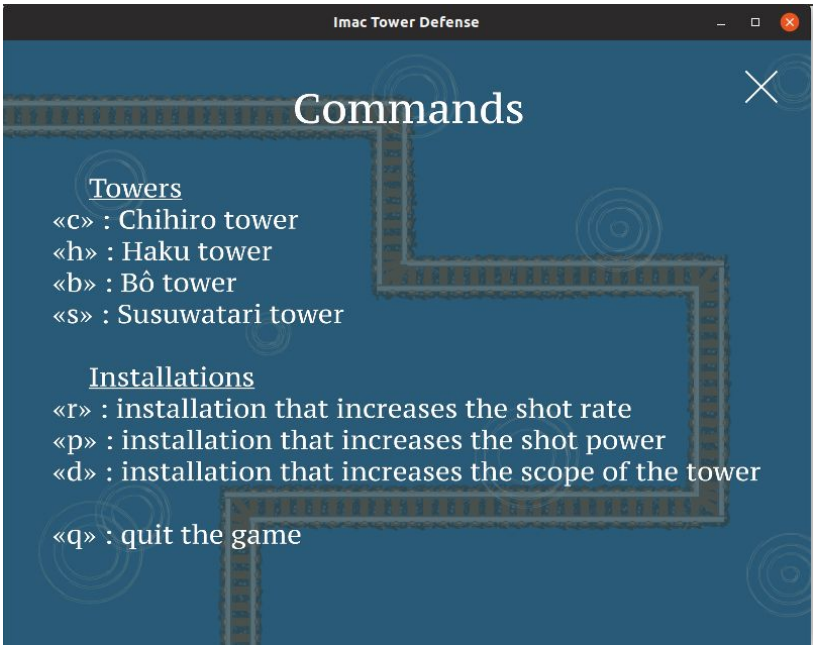
- Ouverture d'une fenêtre SDL de dimension 800*600 pixels sur l'interface qui présente le jeu et propose de commencer une partie ou de consulter les commandes.
- Un fois le jeu lancé, création d'éléments Monstres parcourant la Map par vague de 10. Puis c'est à l'utilisateur de créer les tours et les bâtiments avec des commandes que nous détaillerons par la suite. A l'activation de ces commandes, des tours et des bâtiments sont créés. Les tours peuvent attaquer et tuer les monstres si ces derniers se trouvent dans leur périmètre de tir. Nous n'avons malheureusement pas eu le temps d'implémenter la gestion des bâtiments dans le gameplay, ni la gestion de l'argent.
- Enfin quand une vague de monstre se termine, une nouvelle vague est générée, avec des monstres que nous aurions souhaités de plus en plus puissants, rapides et résistants, mais nous n'avons pas pu implémenter cette fonctionnalité non plus. Ces vagues s'exécutent 50 fois avant la fin du jeu.

6. RÉSULTATS OBTENUS

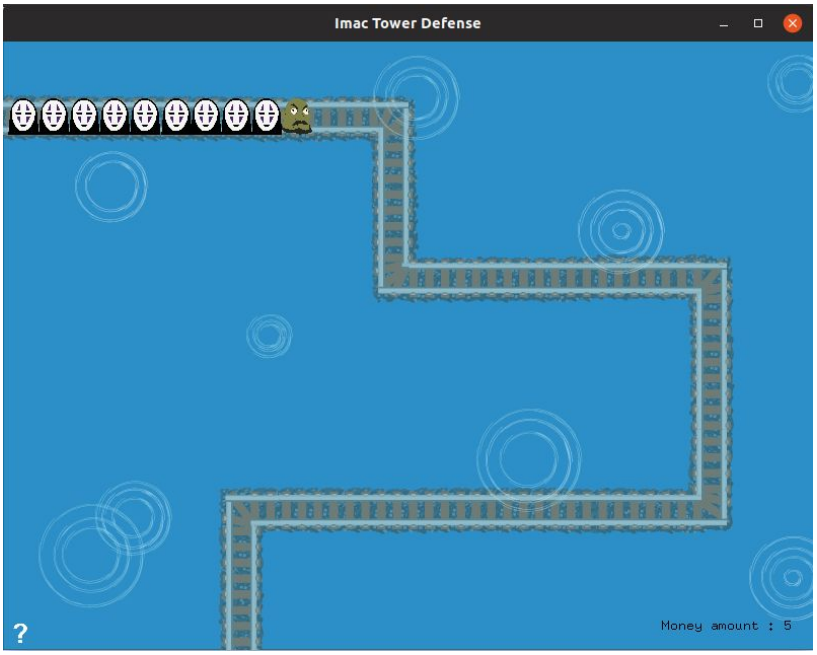
Guide utilisateur :

Voici des captures d'écran des différents résultats que nous avons obtenus :

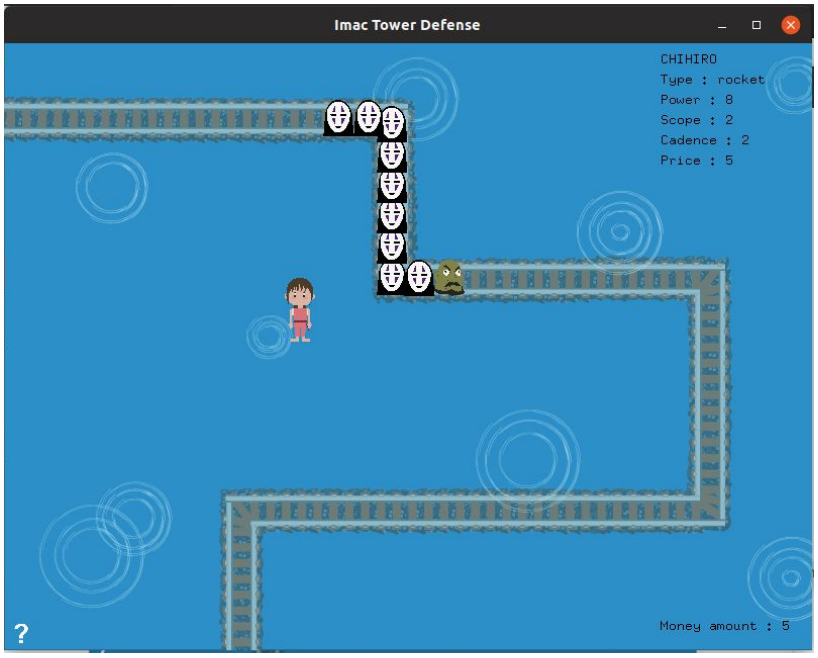
L'utilisateur lance le jeu et arrive sur une page d'accueil. Il peut soit choisir de lancer le jeu, soit choisir de consulter le panneau d'aide. Le bouton "help" lui permettra d'accéder à un affichage lui expliquant comment jouer.



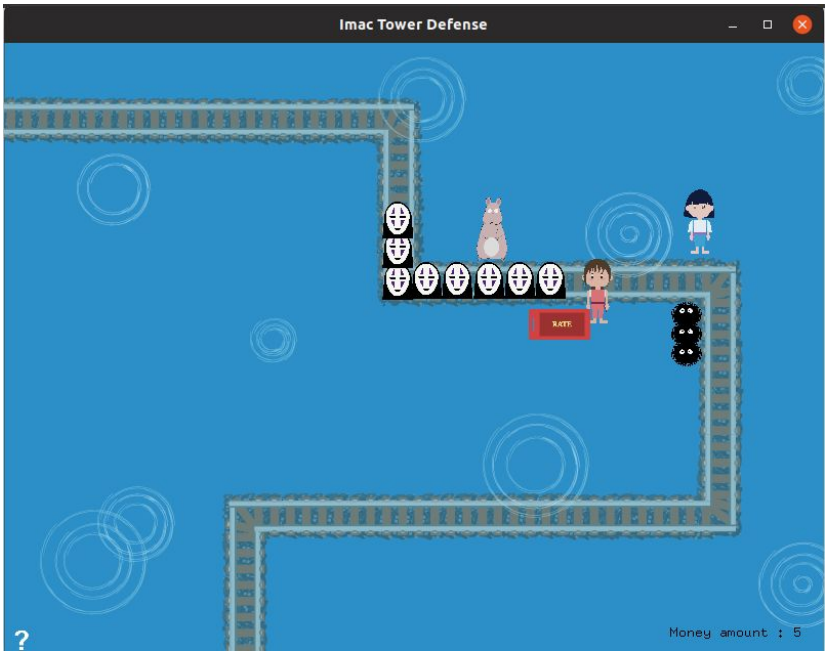
Il peut quitter cette fenêtre en cliquant sur la croix blanche en haut à droite, puis le comportement le plus naturel est d'appuyer sur "start" pour commencer le jeu. En cliquant sur le bouton de gauche il arrive sur le plateau de jeu et le jeu commence.



La première vague de monstres commence immédiatement à apparaître. L'utilisateur peut encore consulter la fenêtre d'aide en cliquant sur le point d'interrogation en bas à gauche de la fenêtre. Il peut à nouveau refermer le panneau à l'aide de la croix située en haut à droite de la fenêtre. La somme d'argent disponible se trouve en bas à droite. Quand il est sur l'interface de jeu, le fait d'appuyer sur différentes touches du clavier (c, h, b, s, p, d, r) lui permet de visualiser les bâtiments qu'ils s'apprêtent à placer. Les informations de ceux ci s'affichent en haut à droite de l'écran.



L’utilisateur peut placer la tour ou l’installation qu’il a sous son curseur en cliquant dans la fenêtre. S’il souhaite changer d’installation sans poser celle sélectionnée, il lui suffit de cliquer sur n’importe quelle touche (sauf q, qui permet de quitter l’application) pour retrouver le curseur normal ou sélectionner une autre installation.



S’il parvient à tuer tous les monstres, la fenêtre reste affichée, attendant l’arrivée de la prochaine vague. S’il ne parvient pas à décimer tous les monstres, la fenêtre se ferme automatique quand le premier d’entre eux parvient à sortir et le message “GAME OVER” s’affiche dans le terminal.

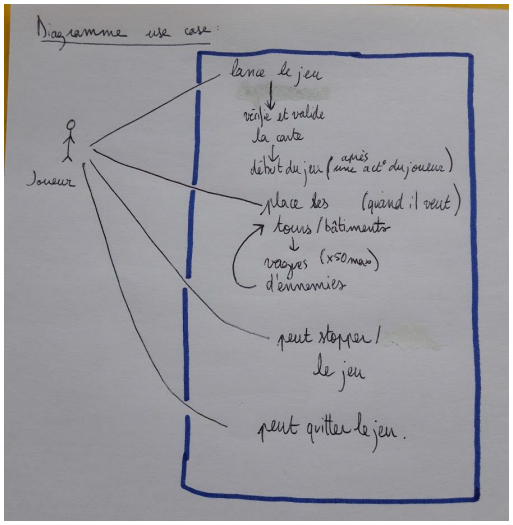
7. RÉPARTITION DES RÔLES

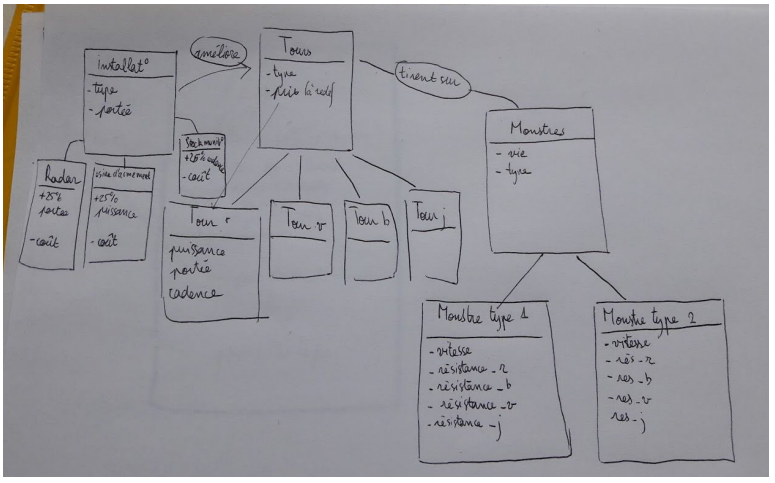
Nous avons décidé de nous mettre ensemble pour ce projet car nous savons que nous avons à peu près le même niveau. Nous voulions progresser toutes les deux, et nous avions déjà travaillé ensemble au premier semestre et cela avait bien fonctionné, nous avons donc décidé de réitérer l'expérience.

Au début du projet, Lucie a commencé par s'occuper de la vérification de la carte .itd et Line de la partie algorithmique concernant les entités et les différentes structures et classes notamment. Nous nous sommes ensuite rendu compte que la vérification de la carte permettait de récupérer les données nécessaires à la réalisation du gameplay et à la manipulation des entités. Les rôles ont donc été modifiés. A partir du milieu du projet et jusqu'à la fin de celui ci, Line, ayant codé les entités, a repris la vérification de la carte et s'est occupée de la gestion des données de la carte par rapport à celles de l'itd en récupérant les données du .ppm. Lucie, elle, s'est occupée la partie SDL et de l'affichage des différentes textures et données récupérées et envoyées par Line, qui est restée concentrée de la partie plus algorithmique.

8. DIFFICULTÉS ET PISTES POUR CONTINUER

La plus grande difficulté que nous avons rencontrée a été la gestion du temps. Comprendre le projet, la logique algorithmique demandée et la manière d'implémenter toutes les fonctionnalités nous a pris beaucoup de temps également. Nous avons en effet longtemps essayé de comprendre la structure du projet que nous allions devoir coder et effectué divers schémas ci-après :





Toute cette période de compréhension et d'appropriation du projet n'a pas été une perte de temps en soi puisqu'elle nous a aidées pour la suite mais elle aurait dû être plus rapide et peut être réalisée immédiatement après la réception des sujets.

De même, l'apprentissage et la recherche de documentation sur ce langage de programmation et de la "bonne" façon de coder a été assez longue. Nous avons pris du retard sur notre programme à cause de notre manque de compétences.

Avec du recul, nous comprenons que nous avons passé trop de temps sur la partie algorithmique. En effet, nous avons été bloqué plusieurs semaine sur la vérification de la carte et du .itd. Finalement, le projet est vraiment devenu enthousiasmant lorsque nous sommes passées à la partie gameplay et que nous avons pu tester les implémentations. Cependant, à cause du manque de temps, nous n'avons pas implémenter tout ce que nous souhaitions et tout ce qui était demandé.

Pour continuer notre projet, nous commencerions d'abord par finir de remplir le cahier des charges initial. Nous voudrions gérer la collision des objets car pour l'instant il est possible de placer des tours et des installations les unes sur les autres, mais aussi sur les chemins.

Nous voudrions aussi améliorer les retours visuels, en affichant les tirs des tours, ainsi que le niveau de vie des monstres, mais aussi les portées des différentes tours et installations.

Il nous paraît également important de pouvoir gérer plusieurs cartes. En effet, dans notre cas nous ne proposons qu'une seule carte à l'utilisateur.

Enfin nous voudrions prévoir des affichages annonçant la victoire ou la défaite de l'utilisateur.

Toutes ces modifications nous paraissent maintenant très faisables. Nous pensons que notre principal problème a été la gestion du temps et pas notre niveau. Si nous avions mieux géré notre temps ou si nous avions eu plus de temps, nous serions sans aucun doute arrivées au bout de ce projet et nous l'aurions fini avec plaisir.

CONCLUSION

Pour conclure, nous souhaitons dire que ce projet nous a beaucoup intéressées et nous a permis d'apprendre beaucoup, tant du côté de l'algorithmique pure que de celui du C++, de celui de l'openGL et de la SDL. Nous sommes un peu frustrées de ne pas l'avoir terminé à temps mais nous nous sentons capable de le continuer et de l'améliorer sur notre temps libre. Malgré cette frustration, nous sommes très contentes de nous-mêmes. Nous ne nous imaginions pas réussir à aller aussi loin quand nous avons pris connaissance du sujet il y a quelques mois de cela.