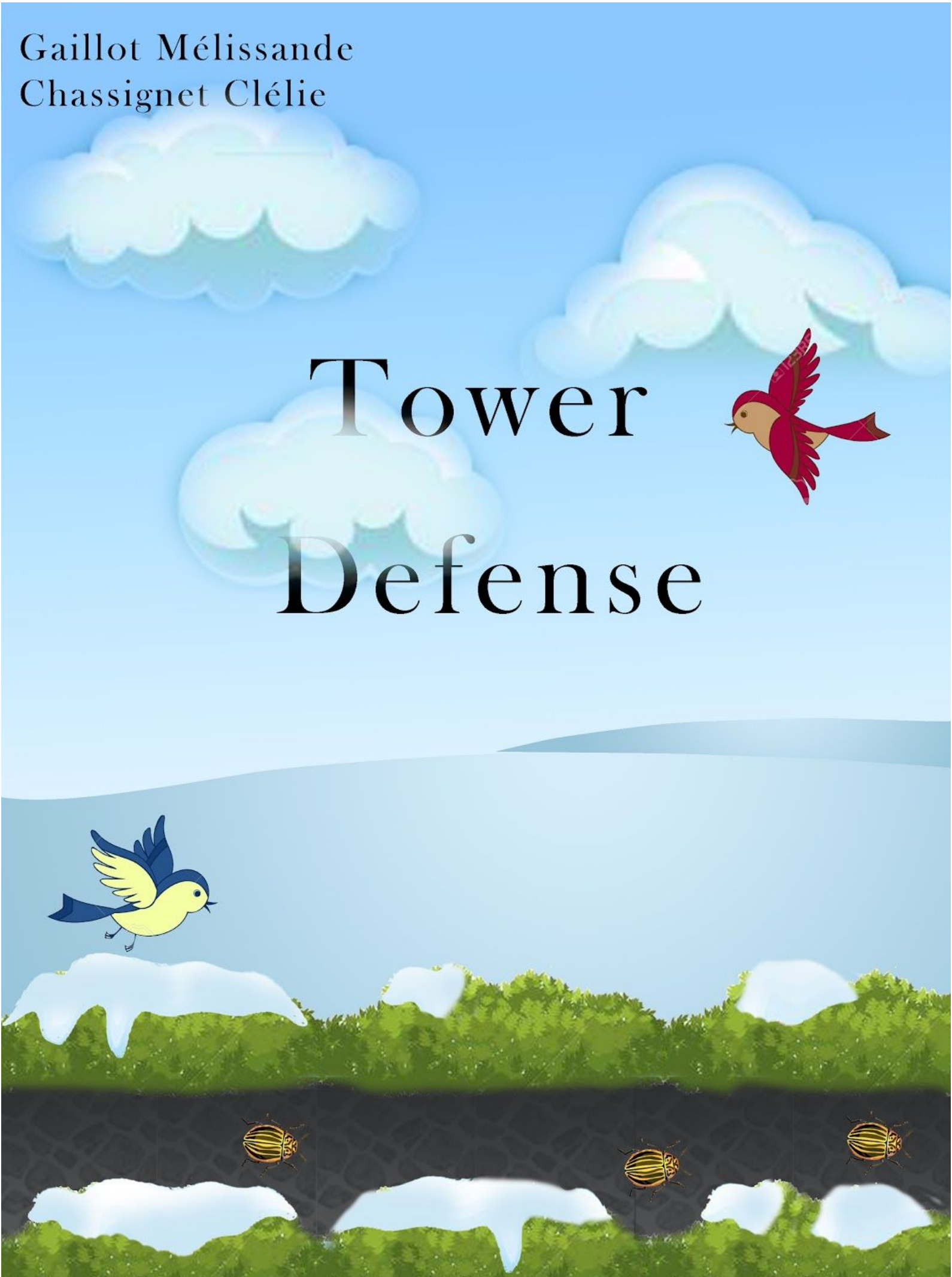
Gaillot Mélissande
Chassignet Clélie

Tower Defense



Introduction

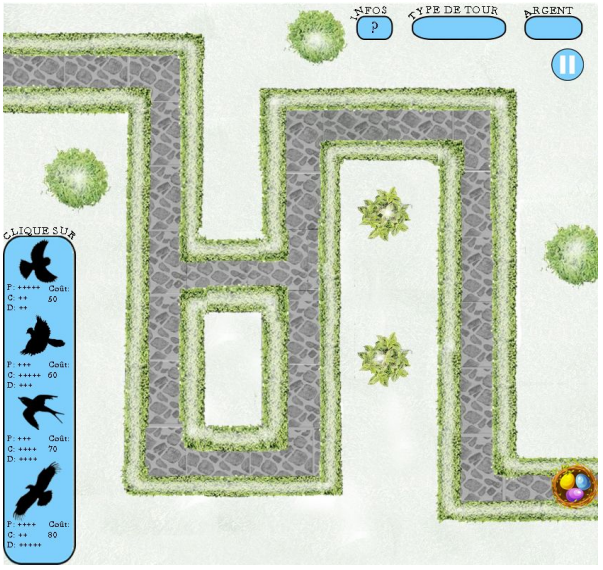
Le projet consiste en la création d'une application affichant une carte sur laquelle des monstres essayent d'aller d'un point de départ à un point d'arrivée en passant par un chemin. Le but est de tuer les monstres en construisant des tours autour du chemin qui leur tirent dessus à une certaine fréquence et avec une certaine puissance, et dont la portée est variable. Des installations peuvent également être construites pour améliorer les tours.

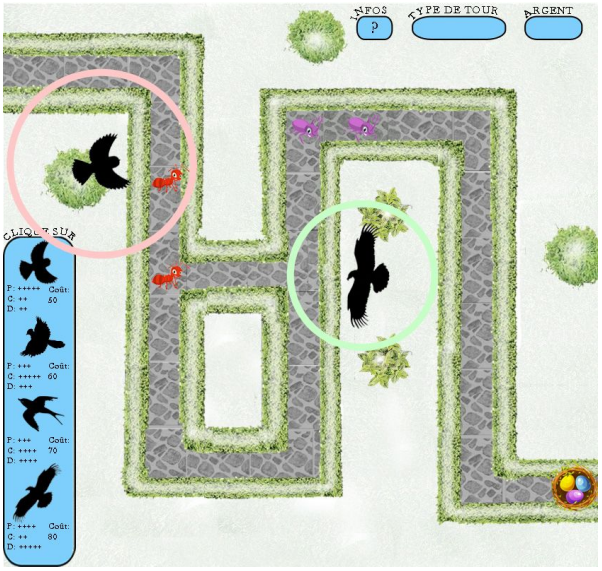
Le projet comprend une partie algorithmique en C++ avec notamment la création du graphe des chemins, la gestion des monstres, des tours et des installations, ainsi qu'une partie infographie avec OpenGL pour afficher la carte et les différents éléments, et gérer leurs actions (tirs, déplacements etc).

1 - Présentation de l'application

La carte

Nous avons imaginé l'univers de notre jeux autour du thème "jardin". La carte de gauche représente ainsi un jardin enneigé avec un chemin de pavés bordé par des buissons. La carte de droite représente

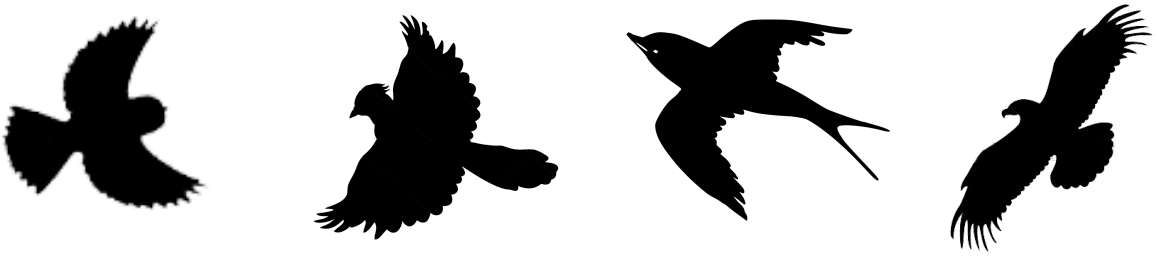




quant à elle une mise en situation avec des tours (les oiseaux) et des monstres (les insectes) en action.

Les tours

Nos tours sont des oiseaux, il peuvent donc aussi bien être placés sur les arbustes que sur le sol. Nous en avons sélectionné quatre que voici dans l'ordre croissant de leur puissance :



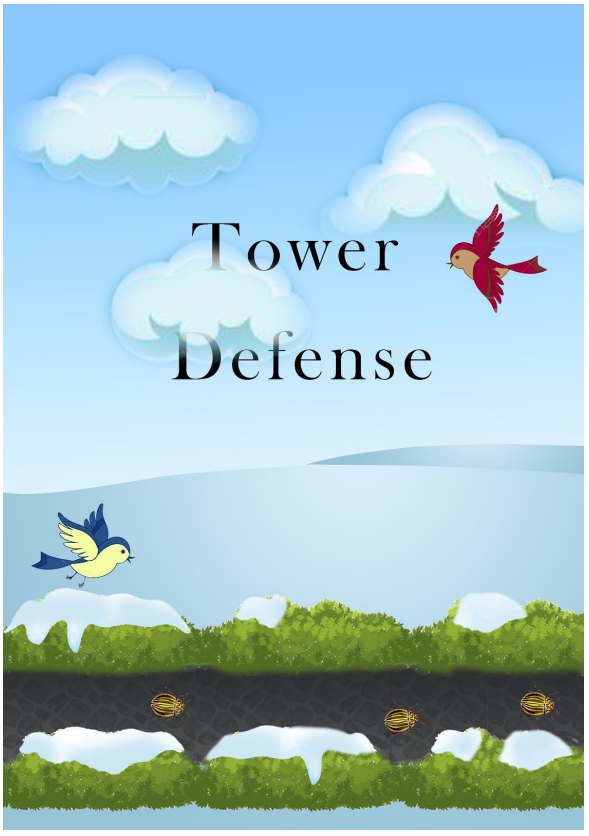
type bleu (merle) type jaune (poule d'eau) type vert (hirondelle) type rouge (aigle)

Les monstres

Nos monstres sont des insectes que les oiseaux doivent détruire pour défendre leur nid. Il y a également quatre types de monstres que voici dans l'ordre croissant de leur puissance :

Les pages annexes

Nous avons ensuite réalisé une affiche pour rendre notre jeux plus attractif, ainsi qu'un menu avec un bouton "commencer une partie" sur lequel il faut appuyer pour arriver sur l'interface avec la carte et commencer à jouer.



Nous avons aussi créé une page "aide" contenant les règles du jeu. Le joueur pourra y accéder en cliquant sur le bouton "Infos". Ainsi qu'une page qui s'affiche lorsque le joueur gagne ou perd. Enfin, une page qui apparaît lorsque le joueur met le jeu en pause.

Pour sauver le nid des oiseaux, décime les quatre espèces d'insectes qui le menacent.
Clique sur le type d'oiseau que tu veux acheter pour défendre le jardin et place-les où tu veux autour du chemin.

Mais surtout n'oublie pas, les insectes ne doivent jamais atteindre le nid !

P=portée C=cadence D=dégâts

Tu as réussi à sauver le nid des oiseaux

Tu as gagné !

Tu n'as pas réussi à sauver le nid des oiseaux

Tu as perdu !

Reprend la partie quand tu es prêt !

Pause

2 - Architecture de l’association

Nous avons commencé par identifier les différentes classes dont nous avons besoin pour le projet, ainsi que les actions de chaque classe :

- Monstres : Affichage, Déplacement, Localisation, Perte de vie et mort,
- Tours : Sélection du type de tour, Vérification de l’emplacement de construction, Affichage, Cercle d’attaque, Tir,
- Bâtiments : Affichage, Pouvoir,
- Carte : Affichage, Charger et lire le fichier .itd, Grille pour la construction,
- Jeu : Menu d’accueil, Vagues de monstres, Constructions de tours et bâtiments, Interfaces utilisateur, Libérer la mémoire

Afin d’éviter les répétitions, nous avons identifié les fonctions communes à plusieurs classes et les avons regroupées dans un fichier “general” avec notamment le chargement d’un fichier image et l’initialisation de la texture pour afficher l’image, ainsi que l’affichage de texte à l’écran.

De plus, nous n’avons pas eu le temps d’implémenter les bâtiments car il nous semblait que les autres éléments étaient plus importants et nous avons donc préféré nous concentrer sur ce qui nous semblait essentiel.

3 - Description des fichiers

3.1 - Le fichier “general”

Dans le fichier “general” nous avons regroupé les fonctions dont nous nous servons à plusieurs endroit.

Il y a tout d’abord la fonction “loadImage” qui prend comme argument le nom du chemin où est stocké l’image en question. On

récupère ainsi toutes les informations de l'image dont nous nous servons en argument d'une deuxième fonction "initTexture". Cette fonction nous permet de créer la texture openGL de type GLuint dont nous avons besoin pour afficher une image.

Nous en avons notamment besoin pour afficher le menu, la carte, les tours, ainsi que les monstres.

De plus, il y a la fonction "afficheTexte" qui, comme son nom l'indique, nous permet d'afficher du texte à l'écran en lui précisant sa position, les caractères à afficher et la police d'écriture. Nous nous en servons par exemple pour afficher l'argent et le type de tours sélectionné.

Enfin, on y trouve la fonction "afficheBoutons" qui prend en argument la texture de l'image à afficher ainsi que les coordonnées que nous souhaitons lui donner. Elle nous permet aussi de décider de la taille de nos boutons. Nous nous en servons pour les boutons "pause" et "play".

3.2 - Le Menu d'accueil

Pour le menu d'accueil, nous affichons simplement une image (grâce aux fonctions décrites précédemment) sur laquelle on peut voir un bouton pour commencer la partie. Si on clique sur la zone où se trouve le l'image du bouton, cela change la valeur d'une variable qui nous permet de lancer tout le reste du programme.

3.3 - La Carte

Les informations permettant de construire la carte sont regroupées dans un fichier de description ITD.
Tout d'abord, il y a le nom du fichier PPM dont nous nous servons pour délimiter les différentes zones de la carte telles que les zones constructibles, les chemins, les noeuds, la zone d'entrée des monstres et la zone de sortie.

Ces différentes zones sont représentées par des couleurs dans le fichier ppm, dont les codes RVB sont indiqués dans le fichier ITD.

Puis il y a le nombre de noeuds que le chemin comporte, et enfin la description de chaque noeud avec : l'indice du noeud, un numéro correspondant à la nature du noeud (1 pour la zone d'entrée, 2 pour la zone de sortie, 3 pour les coudes et 4 pour les intersections). Puis il y a les coordonnées x et y du noeud, ainsi que les indices des noeuds auxquels il est relié.

Nous récupérons ces informations grâce à une fonction "loadITD" dans le fichier "map_reader" afin de connaître la position de chaque noeud ainsi que ses voisins, dont nous nous servons pour coder le déplacement des monstres sur le chemin.

La carte que nous affichons (à l'aide des fonctions du fichier "general" vu précédemment) n'est qu'une image jpg qui reprend le même agencement que le fichier PPM.

Pour simplifier le placement des différents éléments, nous avons une fonction "draw" qui nous permet d'afficher une grille par-dessus la carte afin de pouvoir compter plus facilement le nombre de pixels où l'on souhaite afficher un élément. Cette fonction nous sert uniquement d'aide pendant la phase de développement mais est en commentaire lorsqu'on souhaite jouer afin que la grille ne s'affiche pas.

3.4 - Le graphe des chemins

Les chemins forment un graphe dont les sommets correspondent aux zones d'entrée et de sortie, les coudes et les intersections.

Pour stocker les informations des noeuds nous avons donc créé une structure "mapNode" composée de cinq objets : l'id du noeud, son type (entrée, sortie, coude ou intersection), ses coordonnées x et y et l'id du ou des noeuds adjacents.

Nous stockons ces "mapNode" dans un tableau dynamique de type "vector" que nous avons appelé "nodes". Nous pouvons ainsi récupérer directement toutes les informations de chaque noeud.

3.5 - Les monstres

La gestion des monstres se fait dans le fichier “monster” avec une classe “Monster” qui prend notamment le nombre de points de vies du monstre en fonction du type de monstres ainsi que leur progression dans le graphe des chemins.

Les monstres arrivent par vagues sur le premier noeud du chemin (de type entrée). Les vagues sont gérées directement dans le fichier “main”. La première vague est lancée dès le début du jeu et est composée de seulement trois monstres relativement faibles. Les autres vagues se lancent ensuite automatiquement avec 20 secondes d’écart. Pour chaque vague, on ajoute deux monstres supplémentaires et toutes les deux vagues on met des monstres plus puissants (ils ont plus de points de vie).

Lorsqu’une vague de monstre est déclenchée, il faut les afficher (à l’aide des fonctions du fichier “general”) les uns après les autres en laissant un peu de temps entre chaque monstre afin qu’ils ne soient pas les uns sur les autres (on laisse le programme boucler 60 fois entre chaque monstre).

Puis leur déplacement est géré par la fonction “suivre_chemin” qui détermine à chaque boucle s’il faut incrémenter de 1 la coordonnée x ou le coordonnée y (en fonction de si le déplacement est vertical ou horizontal) et s’il faut l’ajouter ou la soustraire (en fonction de la direction dans laquelle le monstre doit aller pour rejoindre le prochain noeud). Ces valeurs changent donc à chaque fois que le monstre arrive sur un nouveau noeud puisqu’il doit alors changer de direction. On utilise le tableau “nodes” décrit précédemment pour connaître les informations des noeuds et notamment savoir les noeuds adjacents jusqu’à arriver au dernier noeud de type “sortie”.

Puis la fonction “avance” permet d’effectuer une translation avec “glTranslate” dans la direction déterminée précédemment. Ainsi,

l'affichage des monstres est un peu décalée à chaque boucle ce qui donne l'impression qu'ils avancent.

Si un monstre arrive jusqu'à la sortie sans être tué, le jeu est terminé et l'utilisateur a perdu.

La fonction "getCoord" permet de connaître la position d'un monstre à chaque instant grâce à une structure "coord" qui contient les coordonnées x et y du monstre. On en a notamment besoin pour savoir si le monstre entre dans le champs de tir d'une tour qu'on détaillera par la suite.

Si le monstre est touché par un tir, la fonction "degats" lui enlève le nombre de points de vie correspondant à la puissance du tir. Si le nombre de points de vies du monstre arrive à 0, il est mort et donc on le supprime. Cela fait rapporter de l'argent à l'utilisateur qu'il peut utiliser par la suite pour acheter des tours.

Enfin, grâce à la fonction "drawPVMonster", nous affichons la barre de points de vie de chaque monstre au dessus d'eux. Celle-ci représente le pourcentage de points de vie restant. Elle change de couleur lorsque que le pourcentage diminue, c'est-à-dire lorsqu'une tour tire sur le monstre.

Finalement, si le joueur réussit à tuer huit vagues de monstres, il gagne et la fenêtre "tu as gagné" s'affiche. Pour cela on vérifie que huit vagues sont passées mais aussi que le tableau "monsters" est bien vide.

3.6 - Les tours

La gestion des tours se fait dans un fichier "tower" avec une classe "Tower" qui prend la cadence, la puissance, la portée de la tour et les composantes RGB de la couleur de son cercle de portée en fonction de son type.

Pour choisir le type de tours qu'on veut construire on clique dessus dans le menu situé à gauche de la carte. Lorsqu'on clique, un petit cercle apparaît sur l'image de l'oiseau en question afin qu'on soit sûr que notre choix a été pris en compte. De plus, le type de tours sélectionné est écrit

en haut à droite. A côté se trouve l'argent qui diminue du prix de la tour à chaque fois que le joueur en pose une.

Les caractéristiques de chaque type sont stockées à l'aide d'une structure "type_Tower" qui prend la cadence, la puissance et la portée des tirs, ainsi que les composantes de couleur RGB de leur cercle d'attaque et leur coût.

Puis on pose une tour en cliquant sur la carte à l'endroit où on souhaite la placer. Il y a alors une fonction "isConstructible" qui permet de vérifier si l'endroit où on a cliqué correspond à une zone constructible. Pour cela, on a besoin de récupérer la couleur de chaque pixel dans le fichier PPM afin de vérifier qu'il s'agit bien d'une zone constructible (représentée en blanc pour nous). On utilise donc la fonction "readPPM" qui ouvre le fichier en mode lecture et qui stocke les informations RGB de chaque pixel dans un tableau "pixels". Pour cela on utilise une structure "pixel" avec les composantes r, g et b. On regarde donc dans ce tableau si les composantes de couleur du pixel sur lequel l'utilisateur a cliqué correspondent bien à une zone blanche et donc qu'il peut construire à cet endroit.

De plus, on vérifie qu'une autre tour n'est pas déjà à cet endroit car on ne peut pas construire les tours les unes sur les autres.

Si la fonction "isConstructible" renvoie 1 c'est que la tour peut être construite. On soustrait donc son coût à l'argent total. Puis, on enregistre les coordonnées donc un tableau dynamique "coord_towers" ainsi que la texture associée à l'image de ce type de tour dans un tableau dynamique "texture_tower" et les autres caractéristiques de ce type de tour dans un tableau dynamique "element_tower".

Enfin, à chaque boucle du programme, on affiche toutes les tours du tableau avec la fonction "afficheTower" qui utilise la texture créée grâce aux fonctions de "general" vues précédemment. On affiche aussi le cercle de tir de la tour avec la fonction "afficheCercle" dont le rayon dépend de la portée de ce type de tour, ainsi que la couleur.

Pour qu'une tour puisse tirer sur les monstres il faut calculer à chaque boucle la distance de chaque tour avec chaque monstre en

récupérant leurs coordonnées avec la fonction “getCoord” vue précédemment. Si la distance calculée est inférieure à la portée de la tour (stockée dans le tableau “element_tower” avec les autres caractéristiques), cela signifie que la tour peut commencer à tirer sur le monstre. Elle tire donc une première fois, ce qui enlève un certain nombre de points de vie au monstre en fonction de la puissance de tir. Puis il faut attendre un peu avant de pouvoir faire un autre tir si le monstre n’est pas mort et qu’il est toujours dans le cercle de portée de la tour. Ce temps d’attente dépend de la cadence de la tour. S’il y a plusieurs monstres dans le cercle, la tour tire en priorité sur le plus avancé dans le chemin.

Nous affichons le tir avec la fonction “Tir” qui affiche un trait (à chaque fois que la tour tire) qui relie la tour et le monstre en récupérant leurs coordonnées.

4 - Résultats

Nous avons donc réussi à implémenter un jeu fonctionnel avec de jolis graphismes, une interface utilisateur et une partie algorithmique.

On peut lancer une partie, des monstres de plus en plus forts et nombreux arrivent par vagues toutes les vingt secondes, avancent et suivent le chemin en direction de la sortie. On peut ensuite sélectionner un type de tours et les placer autour du chemin sur les zones constructibles si on a assez d’argent. L’argent s’affiche en temps réel en haut à droite.

Les tours tirent sur les monstres ce qui fait baisser leur barre de vie jusqu’à leur mort qui fait gagner de l’argent au joueur. Si un monstre arrive à la sortie, le joueur a perdu. Si le joueur arrive à tuer tous les monstres à la fin des huit vagues, le joueur a gagné.

Au cours de la partie, le joueur peut cliquer sur “info” pour voir les règles ou mettre le jeu en pause.

Nous sommes donc satisfaites de notre jeu au regard de nos connaissances en programmation au départ et au temps imparti.

5 - Les difficultés rencontrées

La première difficulté rencontrée concerne le matériel. En effet, nous n'avons pas Linux sur nos ordinateurs personnels, nous étions donc obligées de travailler à l'université. Seulement, l'université ferme tôt le soir et pendant la journée nous avions cours. De plus, elle est fermée le week-end et les jours fériés, il était donc difficile de trouver des créneaux pour avancer le projet.

L'autre principal problème des ordinateurs de l'école est qu'on ne peut pas installer de librairies, ce qui nous a posé beaucoup de problèmes à plusieurs étapes du projet. Cela est notamment lié au fait qu'on utilise SDL2 et pas SDL, nous n'avons pas pu par exemple mettre de musiques et avons dû trouver des alternatives pour beaucoup d'autres fonctionnalités.

En ce qui concerne le projet en lui-même, nous avons eu du mal à savoir par où commencer car le sujet est très fourni et nous manquions de connaissances en C++ (surtout concernant les classes).

Puis, prises par le temps, nous avons dut accepter d'admettre que nous ne pourrions pas tout implémenter et choisir ce qu'on voulait faire en priorité.

6 - Les améliorations possibles

Nous n'avons pas eu le temps d'implémenter la vérification de la carte, ni l'algorithme Dijkstra pour le déplacement des monstres.

Il nous manque également les bâtiments qui permettent d'améliorer nos tours.

De plus, nous aurions souhaité pouvoir détruire les tours pour récupérer de l'argent.

Nous aurions aussi aimé rajouter de la musique et des bruitages lorsqu'une tour tire, qu'une nouvelle vague arrive ou qu'un monstre meurt.

Conclusion

Au cours de ce projet nous avons beaucoup appris sur la programmation en C++ et particulièrement sur l'aspect orienté objet. Nous avons également pu nous familiariser davantage avec OpenGL.

Il nous a aussi permis d'apprendre à nous adapter aux contraintes matérielles et temporelles, à gérer un projet en binôme et à s'organiser pour le faire en parallèle de tous nos autres cours, partiels et projets.

Nous sommes satisfaites du résultat car nous ne pensions pas arriver aussi loin et nous avons réussi notre objectif d'avoir un jeu fonctionnel et agréable à jouer.