# JEDIMAC
# TOWER DEFENSE

PROJET ALGORITHMIQUE AVANCÉE ET SYNTHÈSE DE L'IMAGE

IMAC 1 – 2018/2019

JULIEN HAUDEGOND & MATHILDE DUMOULIN

# RECAPITULATIF DES TÂCHES

## CE QUI ÉTAIT DEMANDÉ

| TÂCHE | FAIT ? |
|---|---|
| Structure ordonnée avec système de compilation intégré | ✓ |
| Chargement d'une carte | ✓ |
| Création du graphe des chemins | ✓ |
| Vérification de la jouabilité de la carte | ✓ |
| Sprites de bâtiments, de tours, de monstres | ✓ |
| Structures des bâtiments, des tours et des monstres | ✓ |
| Bâtiments et tours :<br>- Ajout<br>- Suppression<br>- Voir stats Tours<br>- Édition | <br>✓<br>✓<br>✓<br>✗ |
| Action des tours<br>- Tirer<br>- Choix de la cible<br>- Résistance des monstres | <br>✓<br>✗<br>✗ |
| Déplacement des monstres | ✓ |
| Gestion des actions des tours et bâtiments | ✓ |
| Gestion du temps | ✓ |

# RECAPITULATIF DES TÂCHES

## CE QUE NOUS AVONS AJOUTÉ

| TÂCHE | FAIT ? |
|---|---|
| Sauvegarde automatique | ✓ |
| Animations : intro (séquence d'images) et fins (animations openGL) | ✓ |
| Charger une carte à partir d'un menu | ✓ |
| Jouer de la musique | ✓ |

# INTRODUCTION

Le but de ce projet était de créer un jeu vidéo, en 2D, vu de dessus, de type “Tower Defense”. Pour cela, nous avons programmé en C et utilisé les librairies SDL et OpenGL, ainsi que FMOD et SDL TTF pour les sons et les textes. En commençant le projet, nous avions surtout en tête l’objectif de nous améliorer et d’étendre le champ de nos connaissances, étant novices en OpenGL. Notre but n’était pas de coder le jeu parfait, mais plutôt de combiner tous les éléments que nous avons appris pour former un ensemble cohérent.

# PRÉSENTATION DE L’APPLICATION

Notre Tower Defense se base sur le thème de la saga Star Wars et les musiques proviennent de la chaîne YouTube 8 Bit Universe. L’application se compose :
- d’une introduction (séquence d’images avec de la musique)
- d’un menu avec différents affichages (sous-menus)
- de la fenêtre principale de jeu

# ARCHITECTURE DE L'APPLICATION

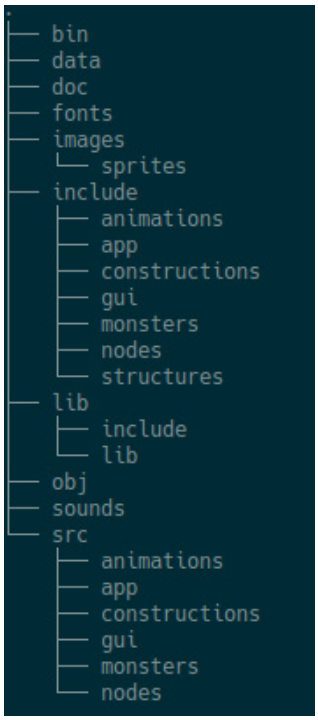
L'architecture de notre projet est découpée comme suit :

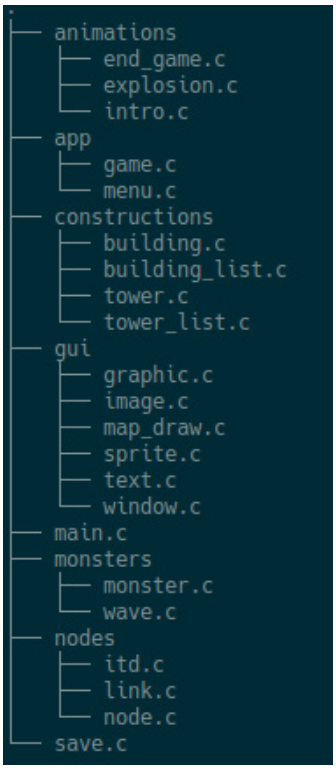
> Headers (prototypes de fonctions & définitions de structures séparés)
> Librairies
> Fichiers sources

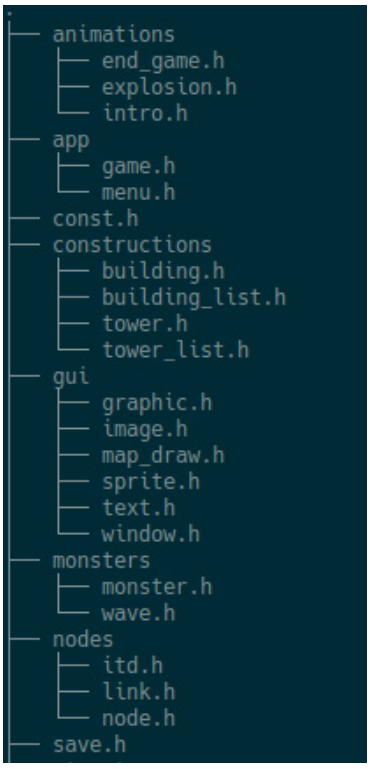
Les headers (prototypes) et fichiers sources sont eux-mêmes répartis selon les dossiers suivants, selon leur fonctionnalité :

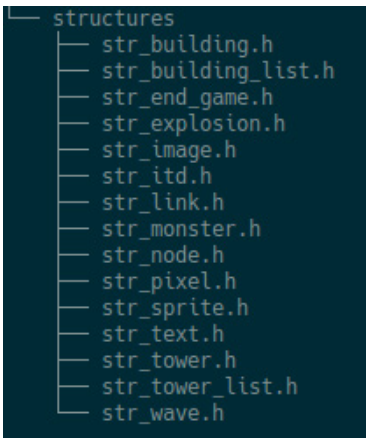
>Constructions (Bâtiments et Tours)
>Nodes (Noeuds, Liens et Vérification de l'ITD)
>GUI (Images, sprites, textes, affichage dans la fenêtre)
>Animations (Introduction, Écrans de fins, Explosions)
>App (Fichier principal du jeu et Menu)
>Monsters (Monstres et Vagues de Monstres)

*Répertoire racine* *Répertoire sources* *Headers / structures*



# FONCTIONNALITÉS INTERACTIVES

## POSER LES TOURS ET BÂTIMENTS :

Le principe même d'un Tower Defense, c'est de poser des tours afin d'empêcher les ennemis de traverser la carte, n'est-ce pas ? Eh bien, pour rendre l'expérience utilisateur encore plus intéressante et plus immersive, nous avons fait en sorte que les cases de notre carte s'allument (en vert ou rouge) au passage de la souris afin d'indiquer au joueur si la case en question est constructible !

## CHOIX DE LA CARTE :

Comme préconisé par le sujet, nous utilisions initialement un argument afin de générer la bonne carte pour la partie. Cependant, on s'est rendu compte que ce n'était pas si compliqué de laisser au joueur le choix de la carte grâce à l'interface graphique. De plus, cela lui permet d'avoir un visuel des cartes auxquelles il va être confronté ! Nous avons donc uniquement chargé une image, qui prend la taille de toute la fenêtre. Si l'on clique sur telle ou telle zone, alors on enclenche un petit algorithme qui va créer le chemin vers la carte correspondante et lancer la partie !

## PAUSE !

Quelle est la fonctionnalité la plus importante dans un jeu ? La pause ! En effet, Jedimac Tower Defense est un jeu qui évolue sans nécessiter la présence du joueur : les vaisseaux continueront d'avancer sur le chemin, quoi que vous fassiez. Il est donc bien pratique de pouvoir stopper le jeu pour vaquer tranquillement à ses occupations !

## SAUVEGARDE AUTOMATIQUE DE LA PARTIE :

Une fois le jeu pleinement fonctionnel, on a souhaité ajouter quelques fonctionnalités... Mettre le jeu en pause n'est pas toujours suffisant si on s'absente longtemps. Nous avons donc implémenté ... La sauvegarde !

Est-ce que ça nous paraissait compliqué et impensable à réaliser avec le si peu de temps qu'il nous restait ? Tout à fait. Cependant, on avait une idée assez précise de comment la réaliser. Après tout, il suffisait seulement de stocker quelques valeurs dans un fichier externe, comme un fichier .txt, par exemple ! Certes, ce n'est pas très sécurisé et des petits malins pourraient facilement trafiquer ce fichier afin de devenir très riches et de finir le jeu en un rien de temps...

Mais au delà de la fonctionnalité, c'est surtout le concept qui nous a paru intéressant à exploiter. Concept qui nous permet de stocker facilement des informations afin de les ré-utiliser plus tard. On aurait ainsi pu faire un classement des joueurs, etc.

```c
int saveGame(const char* itdPath, int money, int wave, TowerList* tl, BuildingList* bl, int status) {
    const char* fichier = "data/save.txt";

    //Open a file
    FILE* F = fopen(fichier,"w");
    if (!F){
        fprintf(stderr, "Unable to open the file.\n");
        exit(EXIT_FAILURE);
    }

    //if the player did not finish the game
    if(status != 0 && status != 1) {

        fprintf(F,"# MAP\n");
        fprintf(F, "%s\n", itdPath);

        fprintf(F,"# MONEY\n");
        fprintf(F, "%d\n", money);

        fprintf(F,"# WAVE\n");
        fprintf(F, "%d\n", wave);

        fprintf(F,"# TOWERS & BUILDINGS\n");
        if(tl->tower) {
            TowerList* tmp = tl;
            fprintf(F, "%d %d %d %d\n", 1, tmp->tower->type, tmp->tower->win_x, tmp->tower->win_y);

            while(tmp->nextTower) {
                tmp = tmp->nextTower;
                if(tmp->tower) {
                    fprintf(F, "%d %d %d %d\n", 1, tmp->tower->type, tmp->tower->win_x, tmp->tower->win_y);
                }
            }
        }
        if(bl->build) {
            BuildingList* tmp = bl;
            fprintf(F, "%d %d %d %d\n", 2, tmp->build->type, tmp->build->win_x, tmp->build->win_y);

            while(tmp->nextBuild) {
                tmp = tmp->nextBuild;
                if(tmp->build) {
                    fprintf(F, "%d %d %d %d\n", 2, tmp->build->type, tmp->build->win_x, tmp->build->win_y);
                }
            }
        }
    }

    //else, the file is overwrite with empty data.

    fclose(F);
    return EXIT_SUCCESS;
}
```

*Algorithme permettant la sauvegarde du jeu*

# CHOIX ALGORITHMIQUES

## S'ASSURER QU'ON EST DANS LE DROIT CHEMIN !

La vérification du fichier .itd était fastidieuse. Il y avait en effet de nombreux tests à effectuer. Afin de construire le graphe des chemins, il fallait vérifier que les noeuds soient bel et bien séparés par des pixels de type “chemin” sur le PPM. Le sujet nous conseillait d'utiliser l'algorithme de tracé des segments de Bresenham. Nous nous en sommes inspirés mais l'avons implémenté un peu différemment, notamment car nous sommes partis du principe que nos chemins seraient uniquement horizontaux ou verticaux.

```
int bresenhamBasedAlgo(Node node1, Node node2) {
    Node min_node;
    int offset;

    //If VERTICAL
    if(node1.x == node2.x) {
        offset = abs(node1.y - node2.y); //Distance between nodes

        //Get the minimum 'y' node
        min_node = node1;
        if(node2.y < min_node.y) {
            min_node = node2;
        }

        //Between 1 and offset because we don't want to check the color of the start node and end node (not a path color)
        for(int i = 1; i < offset; i++) {
            checkCurrentPixelColor();
        }
    }

    //If HORIZONTAL
    else if(node1.y == node2.y) {
        offset = abs(node1.x - node2.x); //Distance between nodes

        //Get the minimum 'x' node
        min_node = node1;
        if(node2.x < min_node.x) {
            min_node = node2;
        }

        //Between 1 and offset because we don't want to check the color of the start node and end node (not a path color)
        for(int i = 1; i < offset; i++) {
            checkCurrentPixelColor();
        }
    }

    //OTHER CASES
    if() {
        //Send errors
    }

    return EXIT_SUCCESS;
}
```

*Pseudo-algorithme de Bresenham*

## CASES DISPONIBLES À LA CONSTRUCTION ?

Comme indiqué précédemment, pour placer les tours et bâtiments, nous devons tester si les cases sont constructibles.
Pour ce faire, nous utilisons la position des tours et des bâtiments et nous parcourons le PPM de la carte afin de vérifier si la case est initialement constructible. Nous avons décidé de faire cela, même si c'est déconseillé par le sujet, car nous avons un PPM de petite taille (20 x 13) qui représente parfaitement notre grille. Qui plus est, c'était plus simple à implémenter ainsi même si ce n'est pas forcément la plus optimisée des façons de procéder.

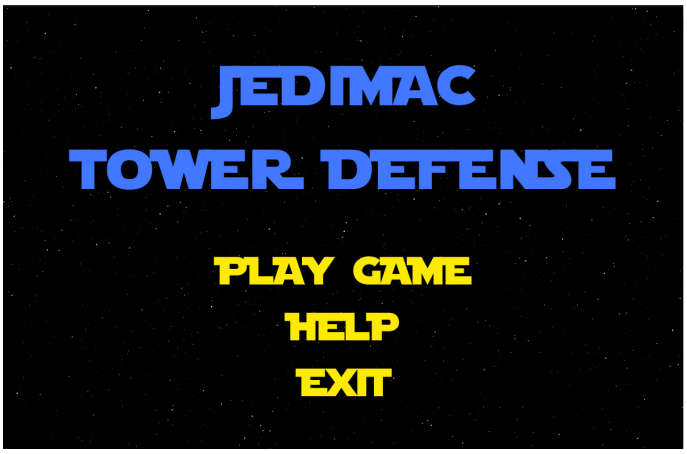
## DES MONSTRES INTELLIGENTS !

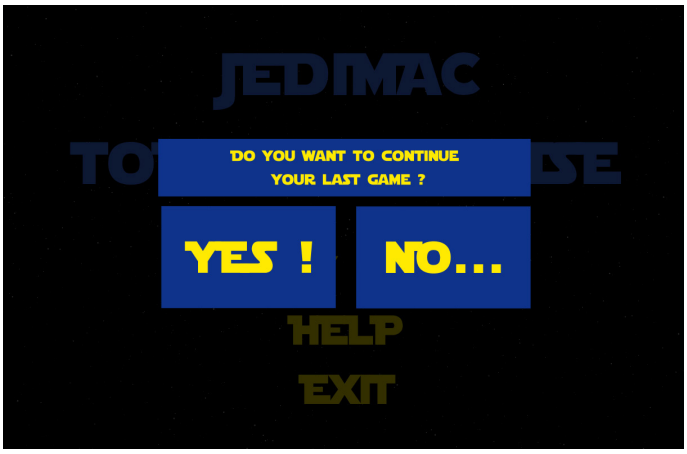
Le chemin emprunté par les vaisseaux ennemis est régi par l'algorithme de Dijkstra. C'est au moment de son apparition qu'on attribue un chemin au vaisseau. Ainsi, son parcours est défini uniquement à cet instant, et n'évoluera pas par la suite. Cependant, l'algorithme est mis à jour lorsqu'une construction est posée. Deux vaisseaux successifs dans une vague peuvent donc avoir une trajectoire totalement différente.
De plus, plutôt que de choisir une entrée aléatoire (lorsqu'il y en a plusieurs), un monstre prendra l'entrée qui fait partie du chemin le plus court, parmi tous les noeuds présents sur la carte. En somme, dans notre cas, l'algorithme de Dijkstra ne prend pas en compte un noeud de départ, ni un noeud d'arrivée, mais l'ensemble des possibilités existantes entre tous les noeuds de départ et le noeud d'arrivée.

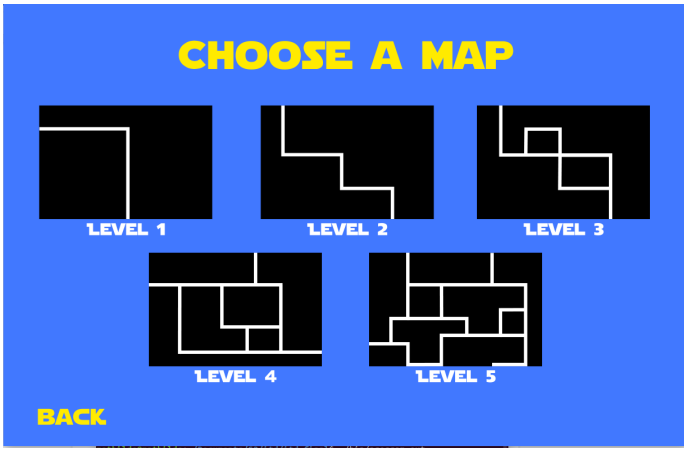
## UNE SEULE VAGUE DE VAISSEAUX ?

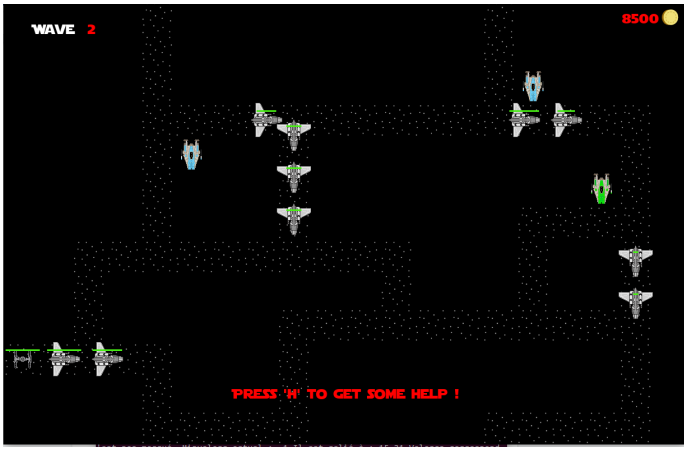
Bien qu'on puisse avoir l'impression qu'il y ait une multitude de vagues (50 !), il y en a en réalité, qu'une seule, plus simple à manipuler pour la mise à jour des coordonnées des vaisseaux et l'affichage. En effet, après avoir créé 10 monstres d'un certain type dans la vague, on incrémente un compteur qui va alors simuler le passage à la vague suivante (de manière visuelle avec le compteur de vagues, mais aussi via le type d'ennemis).

# RÉSULTATS DE L'APPLICATION






# CE QUE NOUS AVONS APPRIS

## BIEN STRUCTURER SON PROJET

Au début de ce projet, nous ne savions pas tellement comment le structurer. Les Design Patterns (que l'on ne connaît pas encore vraiment bien) sont plutôt orientés objet et nous avions décidé de coder en C. On a donc essayé de faire au mieux, comme on le sentait... Et au fur et à mesure du projet, nous nous sommes rendus compte qu'il y avait de nombreuses choses à réorganiser voire à simplifier : en faisant davantage de structures, par exemple. De plus, bien séparer les actions (forte cohésion et faible couplage) permet au programme de mieux fonctionner et au développeur de mieux s'y retrouver, et ça, c'est quand même très important !

## INTÉGRER DES LIBRAIRIES

Ce projet nous a aussi permis d'intégrer le fonctionnement des librairies et de mieux comprendre leurs enjeux dans le cadre d'un projet comme celui-ci. Les termes "statique/dynamique" ou "installée/précompilée" n'ont presque plus de secrets pour nous !
Il est vrai que ce n'était pas simple d'intégrer notre première librairie... des heures de recherches sur StackOverflow et de nombreux conseils de Guillaume sur le sujet nous ont été nécessaires. Et finalement, une fois le concept pris en main, on se rend compte que ça fonctionne toujours de la même façon et que le temps passé sur nos recherches en valait clairement l'investissement.

## INTÉGRER DES ALGORITHMES DÉJÀ EXISTANTS À UN PROJET PERSONNALISÉ

La difficulté principale des algorithmes (déjà existants) à implémenter n'était pas tant leur compréhension, mais le fait de devoir les adapter à un projet précis, répondant à des critères précis. Par exemple, il a été facile de trouver des ressources en ligne afin de comprendre le fonctionnement de l'algorithme de Dijkstra, mais il a fallu l'intégrer au projet en tenant compte de nos structures Node, Links, du fonctionnement des poids... Ce qui fait plein de nouvelles notions à combiner entre elles !

Une des solutions apportées, dans ce cas, est de découper l'algorithme selon les critères suivants :

• Est-ce que je peux rassembler plusieurs traitements pour constituer une "étape" ? Et si ce n'est pas le cas, les lignes restent telles quelles.

• De quelle(s) structure(s) ai-je besoin (pour cette même étape) ?

• Est-ce que je peux grouper certains traitements de l'algorithme dans des fonctions à part ? ( Le problème étant ici d'avoir une cohésion suffisamment forte au niveau du code)

• Ai-je bien pensé à tous les cas de figure possibles ? Cela sert notamment à éviter les erreurs de segmentation par la suite.

## APPRENDRE DE NOS ERREURS

En échangeant avec d'autres groupes, nous nous sommes aperçus que nous pouvons (et de loin) optimiser la boucle principale du fichier game.c, qui contient énormément de lignes de codes, de doublons, d'allers-retours dans des fonctions externes... Par exemple, on aurait pu créer une structure Partie regroupant toutes les informations concernant le déroulement d'une partie : argent du joueur, numéro de vague courant, liste de tours, liste de monstres, liste de bâtiments etc.

# CE QU'ON POURRAIT AMÉLIORER

## LA PAUSE N'INTERROMPT PLUS L'AFFICHAGE

L'ajout de la fonctionnalité "pause" cause quelques problèmes d'affichage. Dans la boucle principale du programme, nous n'avons pas dissocié les fonctions de mise à jour des fonctions d'affichage. Conséquence : quand le jeu est mis en pause, les monstres, tours et bâtiments disparaissent, car les fonctions correspondantes ne sont plus appelées avant le rafraîchissement de l'écran. Pendant la pause, il aurait donc été judicieux de pouvoir stopper uniquement les fonctions de mise à jour en séparant affichage et mise à jour.

## OPTIMISATION DU CIBLAGE DES TOURS

Pour l'instant, les tours ciblent l'ensemble des vaisseaux à leur portée. Il aurait fallu effectuer une variation de l'algorithme de Dijkstra pour savoir quel est le vaisseau le plus proche de la sortie. Ainsi, c'est lui, et lui seul, qui se fait attaquer !

## UN JEU EN MOUVEMENT

Nous aimerions aussi que le jeu soit plus dynamique visuellement, en ajoutant davantage d'animations pour les tours et les bâtiments. Les tours pourraient par exemple tourner en fonction de la direction de leur cible, et les bâtiments seraient animés sur de courtes durées ! Et nous aimerions également rendre l'immersion sonore meilleure avec l'ajout de multiples bruitages (explosions, tirs, écrans de fin, etc).

## UN REPOSITORY PLUS LÉGER

L'idée de faire une introduction nous est venue assez tard. Mais qui dit "Star Wars", dit forcément générique déroulant incliné à 75° selon l'axe X. On ne pouvait donc pas échapper à la règle ! Réaliser cet effet avec OpenGL (ancienne version) semblait assez compliqué, c'est pourquoi nous avons fait une animation vidéo avec Adobe After Effects. Cependant, il nous restait peu de temps avant le rendu du projet et les différentes librairies permettant une lecture vidéo avaient l'air assez complexes... Nous avons donc opté pour une solution plus simple : afficher une séquence d'images (757 images à 25 fps) dans le fond de notre fenêtre. Cela a un coût : 65 Mo de plus dans notre repository... En guise d'amélioration, ce serait donc judicieux d'intégrer une librairie permettant de lire un fichier vidéo compressé qui soit beaucoup plus léger que cette séquence de PNG.

# CONCLUSION

Si ce projet nous a bien appris une chose : c'est que rien n'est impossible. Si quelqu'un l'a déjà fait, alors c'est que nous aussi, on peut le faire ! Patience, autonomie, rigueur et détermination sont les maîtres mots du développeur. C'est en faisant des erreurs que l'on apprend... Et avec ce projet, on a beaucoup appris !

# ANNEXES

Pour mieux appréhender le fonctionnement d'un Tower Defense et avoir une référence, nous nous sommes basés sur le jeu Gemcraft : Chasing Shadows. C'est en observant le déroulement du jeu que nous nous posons ensuite des questions d'ordre algorithmique !


Police de caractères :
https://www.dafont.com/fr/star-jedi.font

Musiques :
https://www.youtube.com/watch?v=Gc4iX80j5HM
https://www.youtube.com/watch?v=4JN8BEP7MjI