

PROJET DU SECOND SEMESTRE

PRESS «SPACE» TO

START

PRESS «Q» TO

QUIT



IMACAT' ATTACK







Sommaire

INTRODUCTION

Thème du jeu

Choix du langage

I. Présentation de l'application

1. Principe du Tower Defense : ImaCat'Attack
2. Division en deux versions
3. Guide utilisateur

II. Description globale de l'architecture

1. Structure globale - arborescence

III. Description des structures de données

1. Diagrammes représentatifs
2. Types de données utilisés

IV. Spécifications techniques

1. Gestion de l'affichage
2. Logique globale du jeu
3. Attaque des monstres
4. Buildings & towers
5. Bresenham

V. Difficultés rencontrées

1. Référence indéfinie
2. Gestion du fichier principal Game
3. Adaptation aux nouveaux langages

VI. Résultats

1. Côté algorithmie
2. Côté affichage

VII. Pistes d'amélioration

1. Organisation
2. Fonctionnalités
3. Expérience utilisateur

VIII. Retour sur expérience

CONCLUSION



Introduction

Dans le cadre de notre projet du second semestre, nous avons réalisé un jeu de **tower defense** pour mettre en pratique ce que nous avons étudié durant le semestre en OpenGL et C++ : **ImaCat'Attack**

L'objectif de cette application était de nous familiariser avec la création d'interfaces en plus de la mise en place d'algorithmes très utilisés dans le domaine de la programmation. Le sujet était composé de deux parties distinctes, une purement algorithmique et l'autre plus axée infographie. Il nous était demandé d'implémenter ces deux parties et de les articuler pour la création de notre jeu.

Nous étions deux pour réaliser ce projet, Eva Benharira et Laurelenn Sangaré.

Thème du jeu

Pour le thème et la réalisation de notre projet, nous souhaitions rendre le jeu de tower defense plus ludique et plaisant. Pour cela nous nous sommes inspirées de la culture internet qui donne beaucoup d'importance au chat, notamment comme source de menace. Le jeu de société Exploding Kitten par exemple, découle de ce modèle.

Il nous fallait ici des monstres : les chats. Mais également une zone à défendre correspondant à la zone de sortie. Pour cela nous avons choisi de gamifier une situation du quotidien. Le joueur incarne un enfant. Celui-ci est chez sa grand-mère et doit empêcher les chats de détruire le vase de cette dernière sous peine de se retrouver accusé.

Sa mission : distraire les chats pour les empêcher d'atteindre le vase grâce aux différents objets mis à sa disposition (qui constituent nos tours).

Choix du langage

Le principe de jeu posé, nous avons pu commencer à coder. Nous avons fait le choix du C++ car ce langage nous semblait être le plus adapté. D'une part parce que Laurelenn était plus habituée à la programmation orientée objet, mais aussi parce que nous nous sommes dit que ce serait un bon entraînement pour les enseignements de l'année 2.

D'autre part, à l'analyse du sujet, il nous est apparu que le jeu était constitué essentiellement d'objets avec divers attributs : les tours, les monstres et les buildings. Ces différents objets possèdent des caractéristiques communes comme des positions x et y. Du fait des notions vues en cours d'architecture logicielle et de nos discussions avec nos camarades nous nous sommes dit que créer une classe abstraite Entité regroupant ces différentes caractéristiques serait une bonne solution, inaccessible en C. Toutes ces raisons nous ont poussées à coder en C++.



I) Présentation de l'application.

1. Principe du Tower Defense : ImaCat'Attack

Comme tout jeu de type tower defense, ImaCat'Attack comprend des zones d'entrée et de sortie des monstres, reliées par des chemins. Autour de ces chemins, les joueurs peuvent construire des tours, ici des distractions, qui détruisent les monstres pour les empêcher d'atteindre la zone de sortie. Les joueurs peuvent également créer des buildings pour améliorer les statistiques de leurs tours. Cela seulement si leurs cagnottes, alimentées par la disparition des chats, le leur permettent !

Dans notre jeu, les chats sortent de leur panier et suivent un chemin à travers le salon jusqu'à l'objet du délit : **le vase**. Le vase constitue ici la zone de sortie et donc, pour le joueur, la zone à défendre.

Si le joueur survit à 20 vagues de monstres, c'est gagné ! S'il casse un vase avant la fin de la vingtième vague, c'est perdu pour lui et l'enfant se fera gronder par sa grand-mère !

2. Division en deux versions

Le jeu ImaCat'Attack devrait permettre de lancer une partie, de construire des tours qui tirent sur les chats lorsqu'ils passent à leur portée. On devrait également pouvoir construire des buildings, soit des recharges pour nos distractions, qui permettent de modifier les caractéristiques des tours, augmentant leur portée, leur cadence ou leur puissance.

Cependant, si nous avons réfléchi à ces différentes fonctionnalités et tenté d'en implémenter une partie, nous nous sommes retrouvées confrontées à des erreurs qui ne nous permettent pas d'exécuter véritablement notre programme. La prise de conscience de cette difficulté nous a poussé à développer une deuxième version de code, simplifiée, gérant principalement l'affichage.

Il nous aurait fallu en effet plus de temps pour déboguer complètement notre première version et comprendre l'erreur à laquelle nous avons été confrontées ([voir partie V.1.](#)). C'est pourquoi nous avons choisi de recréer une version simplifiée mais qui fonctionne, afin de pouvoir certifier et prouver le fonctionnement de la partie affichage et de la création de la map et gestion des zones constructibles.

Nous présentons donc deux versions de code. L'une dans laquelle nous avons poussé la réflexion algorithmique sur de nombreux points mais que nous ne parvenons pas à exécuter et une seconde version qui ne possède pas une véritable structure mais nous permet de vous présenter diverses fonctionnalités d'affichage et de gestion de la map.



3. Guide Utilisateur :

Objectif : Vous devez empêcher les chats de renverser le vase de votre grand-mère. Pour ce faire, vous avez plusieurs sources de distraction à votre disposition : les tours.

3.1. Les tours :

Comment construire une tour ?

Pour actionner la fonctionnalité “construire une tour”, il vous suffit de presser la **touche “T”** puis d’appuyer sur **la touche du chiffre** correspondant au **type** de tour que vous souhaitez (celui-ci est initialisé à 1 par défaut).

Touche 1 : la tour **RED_LASER** - un jeu laser

Touche 2 : la tour **GREEN_GRASS** - de l’herbe à chat

Touche 3 : la tour **YELLOW_GAMMELLE** - La gamelle de croquettes

Touche 4 : la tour **BLUE_MILK** - un bol de lait

Une fois que cela est fait, **cliquez sur la zone** où vous voulez construire votre tour. Bien sûr, vous ne pouvez pas construire sur le chemin des monstres ! Cliquez à nouveau sur la **touche “T”** pour désélectionner la fonction de construction.



Chaque tour a une portée, un prix, une cadence de tir, et une puissance d’effet différents. Vous pouvez placer une tour par case constructible tout en ayant conscience que celles-ci ont une portée limitée. **Mais vous pouvez améliorer les tours et leurs caractéristiques grâce aux buildings !**

3.2. Les buildings :

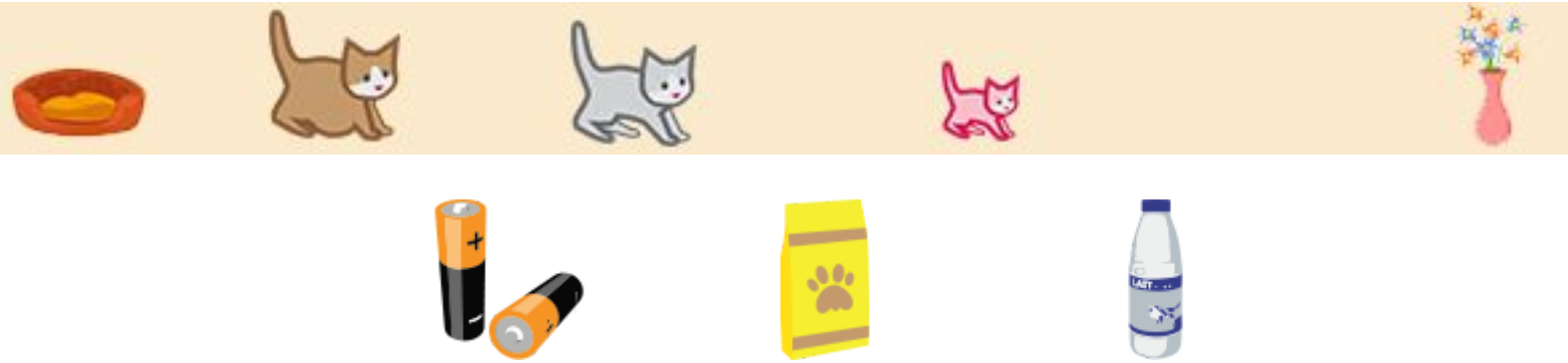
Comment construire un building ?

De même que pour les tours, pressez la **touche “B”** pour activer le mode “construction de building”. Il existe trois types de “recharges” pour les tours qui correspondent aux différents buildings, à même d’en modifier les caractéristiques. Pressez **la touche du type correspondant** pour choisir :

Touche 1 : le building **RADAR** - augmente la portée des tours dans son rayon d’action.

Touche 2 : le building **WEAPON** - augmente la puissance des tours dans son rayon d’action.

Touche 3 : le building **STOCK** - augmente la cadence des tours dans son rayon d’action.



Pressez à nouveau la **touche “B”** pour désactiver le mode “construction de buildings”.

Attention !

Si vous n’avez pas assez d’argent dans votre cagnotte, vous ne pourrez pas construire vos tours ou vos buildings !

3.3. Les Chats :

Vous pouvez rencontrer trois types de chats :

- **Les Kitten** : Rapides, mais peu résistants aux distractions.
- **Les FatCat** : Lents, mais très résistants aux distractions.
- **Les JustCat** : D’une rapidité et d’une résistance moyenne.

Il y a 10 chats par vague, plus le niveau est important, moins il y a de JustCat !



3.4. Quitter le jeu :

Pour quitter le jeu, il vous suffit de presser la **touche “Q”**, la **touche “ESC”** ou de **cliquer sur la croix** !
Votre progression ne sera pas sauvegardée !



II) Description globale de l'architecture

1. Structure globale - arborescence

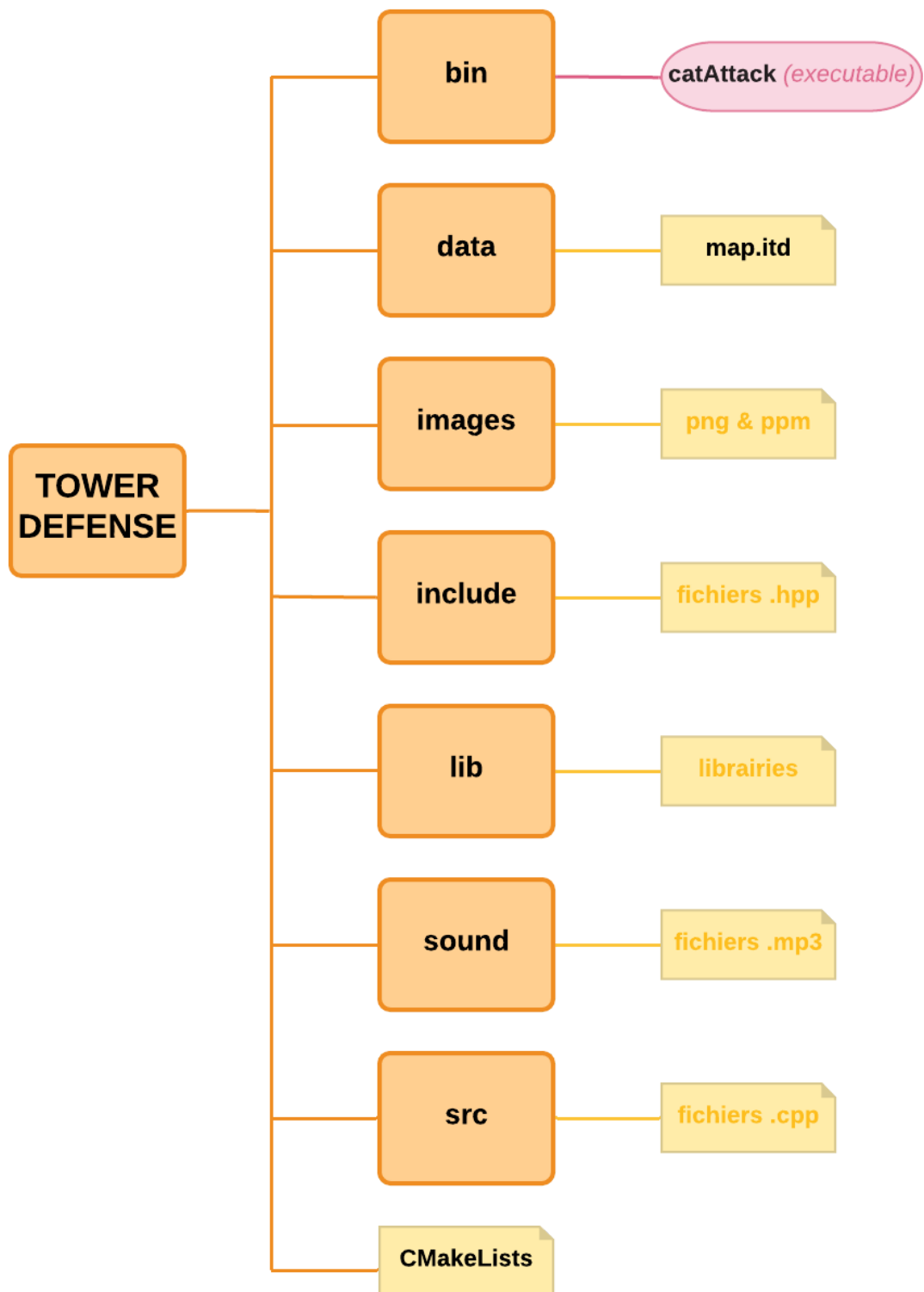
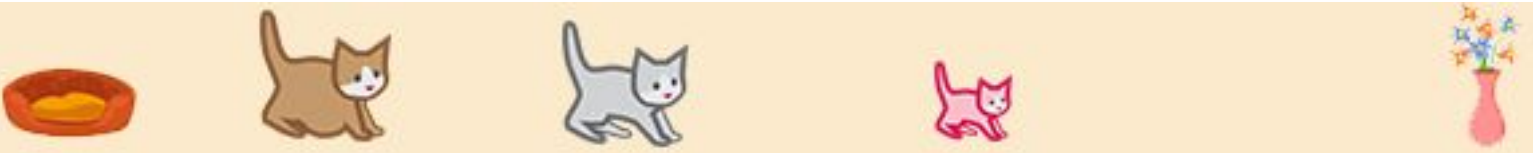
Comme nous avons pu vous l'expliquer précédemment, il n'y a pas un dossier principal TowerDefense mais deux : Un étant très complet sur l'algorithmique mais dont les erreurs nous empêchaient d'afficher quoi que ce soit, et un second plus léger algorithmiquement parlant mais qui permettait de montrer le potentiel visuel du jeu.

Dans les deux cas, leur arborescence individuelle ne change pas : Nous nous sommes fortement inspirées de l'arborescence recommandée dans le sujet, ainsi que de celles créées dans nos anciens projets.

Ainsi, la répartition des fichiers est la suivante :

Le **dossier principal (TowerDefense)** comporte les dossier **bin**, **data**, **images**, **include**, **lib**, **sound**, **src**, ainsi qu'un **CMakeLists** (qui nous permet de générer un MakeFile).

Bin contient notre exécutable nommé "catAttack", **data** contient notre fichier itd "map.itd", **include** contient nos fichiers en .hpp, **lib** contient nos librairies (*voir III.2*). **Sound** est un dossier contenant des fichiers .mp3 qui pourront être ajoutés à l'avenir dans une v3, enfin, **src** contient nos fichiers .cpp dans lesquels se trouve tout notre algorithme.



Arborescence du projet ImaCatAttack.



III) Description des structures de données utilisées

1. Diagrammes représentatifs

Dans les deux versions de notre ITD, la structure de données repose sur quatre grandes classes : Les tours, les buildings, les monstres et la map. Il y a également la classe "tile" qui nous permet de définir les zones constructibles et non constructibles, ainsi que de savoir où sont l'entrée, la sortie, et les nodes de la map. Aussi, nous avons créé des fichiers gérant l'affichage des textures ou l'initialisation de la map par la lecture de l'ITD.

Dans les deux cas il est important de les appeler dans une fonction principale (main ou game) afin de pouvoir articuler ces éléments entre eux.

1.1 Côté Affichage

Dans notre version simplifiée du jeu, nous nous sommes concentrées sur les classes essentielles à l'affichage de celui-ci :

Les classes Map, Tile, CatMonster, Tower et Building; ainsi que les fichiers gérant la texture et la lecture des fichiers ITD et PPM.

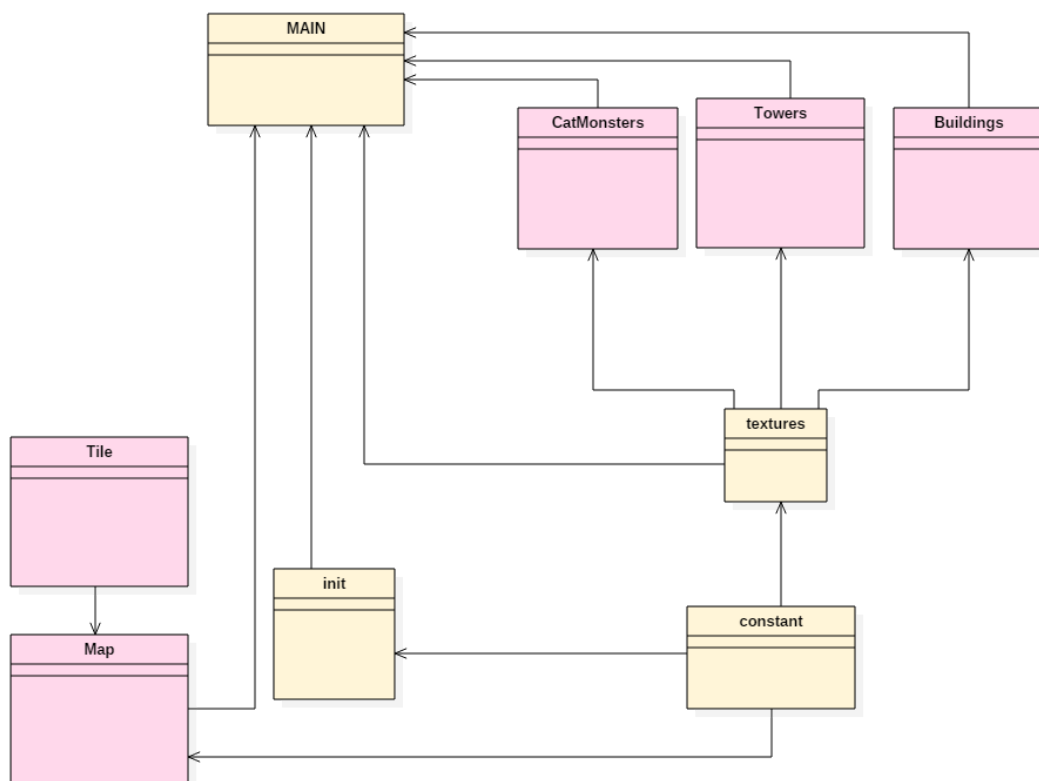


Diagramme de l'ITD simplifié.



Dans le cas présent, nous pouvons voir que toutes les grandes classes (ainsi que le fichier init) sont liées au fichier main qui gèrent le gameplay et les événements du jeu. Nous remarquons également l'inclusion des fichiers essentiels à l'affichage des entités et de la fenêtre de jeu (constant, texture).

1.2. Côté Algorithme Complexe

La réflexion sur cet algorithme n'a pas été facile au début, bien que quelques éléments nous soient apparus comme évidents, notamment l'utilisation de nos quatre classes principales comme notifié ci-dessus. Nous avons au début directement pensé à utiliser une classe abstraite "Entité" dont hériteraient Tower, Building et CatMonster étant donné leurs attributs communs.

Nous avons également créé une classe map, et les classes s'articulant autour de celle-ci : La classe Tile dont nous avons parlé précédemment, mais surtout la classe Node. Les nodes devaient être créés en fonction de la lecture de l'ITD, puis nous devons vérifier la cohérence entre les données de l'ITD et du PPM, pour ensuite inclure la Tile associée dans les attributs du node.

Evidemment, nous devons pouvoir retrouver la Tile associée à une position x,y de la map, qui permettrait notamment aux monstres de se déplacer et aux tours et buildings d'être construits.

La classe game elle, prend en charge la partie en elle-même : les différentes actions de l'utilisateur et réactions du jeu. Cette classe permet de gérer la cagnotte, argent dont dispose le joueur, sa diminution lorsqu'il achète et construit des tours, et son augmentation lorsque des monstres meurent. Elle gère également le nombre de vagues et de ce fait stocke dans des vecteurs les monstres, tours et buildings présents à l'écran et donc appartenant à la partie à l'instant t. Il était important que les entités puissent accéder aux vecteurs des unes et des autres, c'est pourquoi elles ont un attribut pointeur vers leur Game. La création d'une partie et son lancement est créée dans le fichier main.

Nous avons à nouveau un controller de textures qui nous permet de charger nos images et textures. Il y a également un fichier window qui nous permet de créer notre fenêtre de jeu.

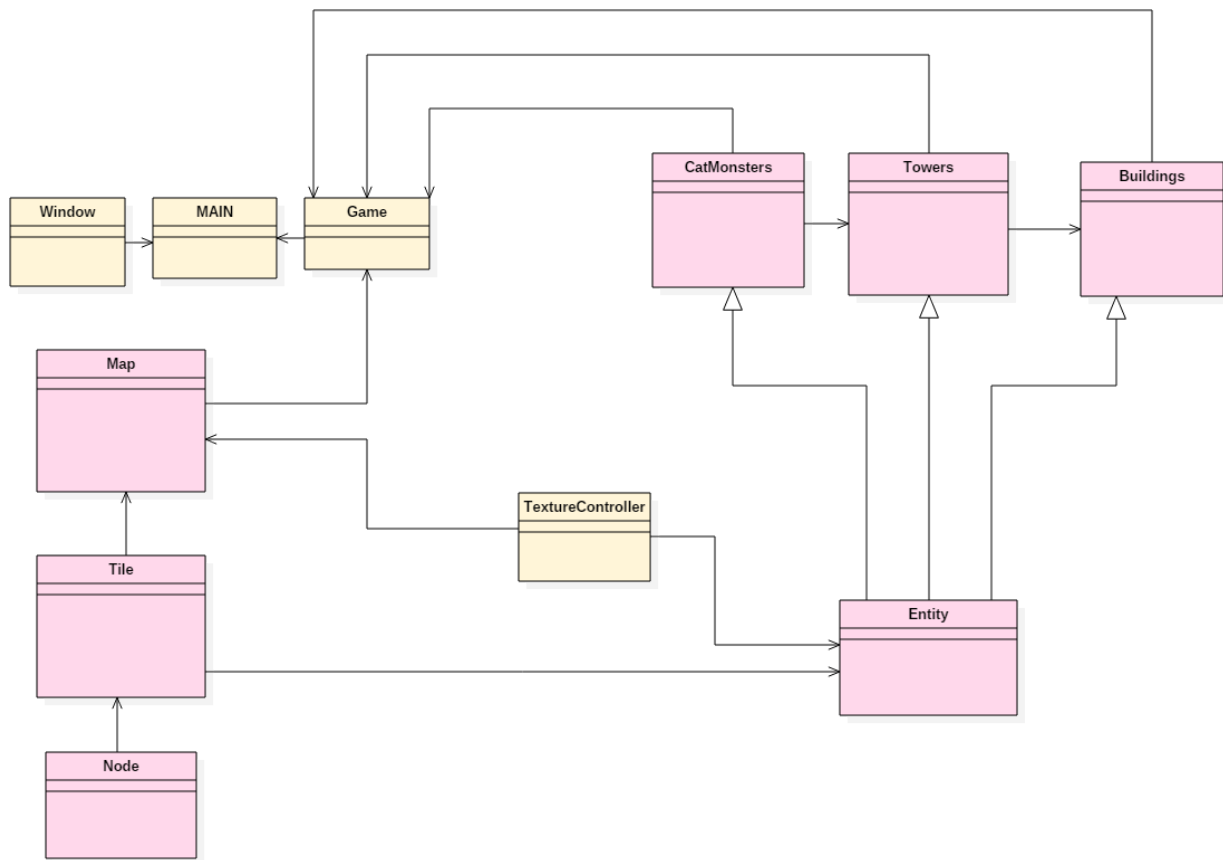


Diagramme de l'ITD complexe.

Bien que la seconde structure de données soit plus complète, nous avons pu prendre du recul lors de la réalisation de la version simplifiée sur celle-ci. En effet, il n'était peut-être pas nécessaire de diviser en tant de fichiers différents nos fonctionnalités (par exemple les fichiers Game ou Window). Cela permet néanmoins de comprendre plus facilement l'architecture et l'utilisation de nos fichiers, notamment pour les personnes extérieures.

2. Types de données utilisés

Concernant les types de données utilisés, nous avons créé beaucoup de structures/classes (Game, Entité, CatMonster, Tower, Building, Map, Node, Tile pour la version complexe).

La version algorithmique est extrêmement complexe au niveau des inclusions et des pointeurs entre les classes, et c'est sans doute à cause de cela que nous nous sommes retrouvées bloquées au bout d'un moment. En effet, la map contient une liste de node et de cases, les pointeurs sur cases étant eux-mêmes des attributs de node, etc.

Le deuxième point important à soulever dans nos deux versions du Tower Defense est l'utilisation de vecteurs qui a grandement simplifiée le développement de notre projet. En effet, nous utilisons ce vecteur pour savoir combien il y a d'entités sur la map, cela est très pratique lorsqu'il s'agit de toutes les parcourir, pour trouver une cible à la tour, ou upgrade les tours à proximité des buildings par exemple. C'est sans aucun doute le type de données ayant le plus d'importance dans la logique de notre projet.



IV) Spécifications techniques

1. Gestion de l'affichage

La gestion de l'affichage concerne principalement la V2 étant donné qu'elle a pu être testée sur celle-ci. Dans la fonction principale main, on actualise à chaque boucle (tant que le jeu est en cours) la texture des entités. (voir IV.2.)

Nous avons vu sur notre V2 que cela fonctionnait parfaitement avec les tours et les buildings, et il est très facile de l'appliquer aux monstres (la fonction étant déjà créée) et à leur déplacement. Puisque leur texture est mise à jour continuellement en fonction de leur position, l'image du monstre avancera en même temps que celui-ci.

```
void Tower::update(const char path[]){

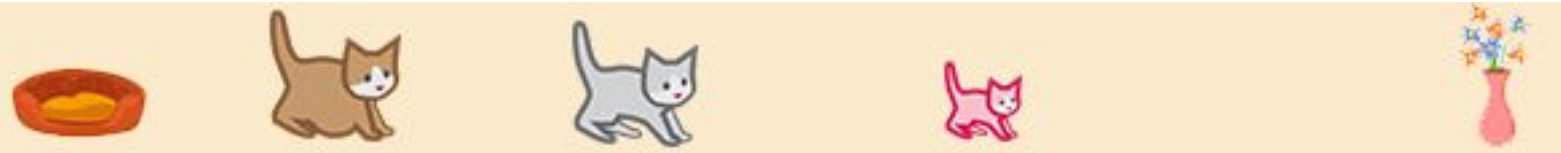
    // Draw collision circle
    float segments = 16.0f;
    float increment = 2.0f * M_PI / segments;
    float theta = 0.0f;

    glColor4f(0, 0, 0, 0.2);
    glBegin(GL_POLYGON);
    // glBegin(GL_LINE_LOOP);
    for (int i = 0; i < segments; i++) {
        float x = m_x + m_radius * cos(theta);
        float y = m_y + m_radius * sin(theta);
        glVertex2f(x, y);
        theta += increment;
    }
    glEnd();

    ///// CHARGER TEXTURE /////
    float x = this->getX();
    float y = this->getY();
    GLuint towerTexture;
    towerTexture = initTexture(path);

    // glClearColor(GL_COLOR_BUFFER_BIT);
    glColor4f(255, 255, 255, 1);
    glBindTexture(GL_TEXTURE_2D, towerTexture);
    glBegin(GL_QUADS);
    glTexCoord2f(0, 1); glVertex2f(x-25.0, y+25.0); // bas gauche
    glTexCoord2f(1, 1); glVertex2f(x+25.0, y+25.0); // bas droite
    glTexCoord2f(1, 0); glVertex2f(x+25.0, y-25.0); // haut droite
    glTexCoord2f(0, 0); glVertex2f(x-25.0, y-25.0); // haut gauche
    glEnd();
    glBindTexture(GL_TEXTURE_2D, 0);
}
```

Tower::Update(filename)



La fonction update pour les tours et les buildings est quasiment identique, nous voyons en premier qu'elle dessine un cercle du rayon de la portée de la construction, afin de bien visualiser jusqu'où les constructions ont de l'effet. Ensuite, nous appelons la fonction initTexture() qui charge une texture en fonction du chemin donné en paramètre d'update (glGenTextures(), glBindTexture() etc...).

2. Logique globale du jeu

Dans les deux versions de notre jeu, nous utilisons les vecteurs afin de pouvoir récupérer plus facilement les entités de notre jeu. On s'en sert, dans la version 2, pour actualiser l'affichage de nos tours, mais aussi pour les supprimer une fois le jeu terminé.

```
std::vector<CatMonster*> catMonsters;  
std::vector<Tower*> towers;  
std::vector<Building*> buildings;
```

Initialisation des vecteurs d'entités

```
////////// UPDATE ENTITES //////////  
for (Tower* tower : towers) {  
    if(tower->getType()==RED_LASER){  
        tower->update(tower_red_path);  
    } else if(tower->getType()==GREEN_GRASS){  
        tower->update(tower_green_path);  
    } else if(tower->getType()==YELLOW_GAMMELLE){  
        tower->update(tower_yellow_path);  
    } else if(tower->getType()==BLUE_MILK){  
        tower->update(tower_blue_path);  
    }  
}  
  
for (CatMonster* catMonster : catMonsters) {  
    catMonster->update();  
}  
  
for (Building* building : buildings) {  
    if(building->getType()==RADAR){  
        building->update(building_radar_path);  
    } else if(building->getType()==WEAPON){  
        building->update(building_weapon_path);  
    } else if(building->getType()==STOCK){  
        building->update(building_stock_path);  
    }  
}
```

Parcours des vecteurs pour update les affichages



```
for (Tower* tower : towers) {  
    delete tower;  
}  
for (CatMonster* catMonster : catMonsters) {  
    delete catMonster;  
}  
for (Building* building : buildings) {  
    delete building;  
}
```

Destruction des éléments dans les vecteurs

Quand à la V1, les vecteurs sont très utiles pour les fonctionnalités ci-dessous.

3. Attaque des monstres

L'attaque des monstres par les tours nous a posé problème pendant un moment. Notamment pour que le monstre, une fois mort, disparaisse du vecteur de monstres ainsi que pour que le jeu récupère le gain du monstre.

C'est la raison pour laquelle catMonster peut accéder à Game grâce à un pointeur sur celui-ci dans ses attributs.

La deuxième question que nous nous sommes posées était celle de la cible. Comment dire à la tour quel monstre doit-elle viser ? Pour cela, la tour, grâce à un pointeur vers Game également, accède au vecteur de monstres, et sélectionne comme cible celui qui est le plus proche ET dans sa portée. Tant qu'elle n'a pas trouvé de cible, elle n'attaque pas.

Lorsqu'elle attaque un monstre, celui-ci est endommagé, on vérifie alors s'il est vivant ou mort. S'il est mort, on le retire de la cible de la tour (raison pour laquelle la fonction beDamaged() prend le pointeur sur tour en argument), il donne le gain à la cagnotte du game, et se retire du vecteur.



```
while(this->getTarget()==nullptr){
    this->searchTarget();
}
printf("TIIIR !! ");
this->getTarget()->beDamaged(this->getDamages(), this);
startTime = SDL_GetTicks();
}
}
}

void Tower::searchTarget() {
    int portee = this->getPortee();
    float distanceMin = portee;
    CatMonster* catClosest = nullptr;
    Game* game = this->getGame();

    for (CatMonster* cat : game->getVecCat()) {
        Tile* cTile = cat->getTile();
        float distanceCurrent = cTile->distance(this->getTile());
        if (distanceCurrent < distanceMin) {
            distanceMin = distanceCurrent;
            catClosest = cat;
        }
        this->setTarget(catClosest);
    }
}
```

Fonctions Tower::attack() & tower::searchTarget()

```
void CatMonster::beDamaged(int nbDamages, Tower* tower){
    setLife(this->getLife()-nbDamages); //retire nbdégats reçus à nbPV possédés
    if (!this->isAlive()){
        this->destroy(tower); //détruit monstre si n'a plus de vie
    }
}

bool CatMonster::isAlive(){
    return this->m_life > 0;
}
```

Fonction CatMonster::beDamaged



4. Buildings & towers

Les vecteurs sont également utiles, dans la V1, pour déterminer si un building influe sur une tour ou pas.

En effet, nous avons créé dans building une fonction `upgradeTower()` qui prend en argument une tour qui sera améliorée en fonction du type du building. Cela nécessite une double vérification : une au moment où nous construisons une tour, et une lorsqu'il s'agit d'un building.

Pour cela, dès que nous construisons une tour, nous vérifions si elle est dans la portée d'un des buildings du vecteur de buildings. Si c'est le cas, elle se fait upgrade par chacun des buildings dont elle est à la portée. De même, lors de la création d'un building, il va upgrade chaque tour qui est à sa portée en naviguant dans le vecteur de tours.

Bien sûr, avant de construire une tour ou un building, nous vérifions que la zone soit bien constructible et que la cagnotte soit suffisamment approvisionnée.

```
void Game::checkTowers(Building *b){
    int portee = b->getPortee();

    for (Tower* tower : this->getVecTower()){
        Tile *tTile = tower->getTile();
        float distance = tTile->distance(b->getTile());
        if (distance <= portee){
            b->upgradeTower(tower);
        }
    }
}

void Game::checkBuildings(Tower *t){

    for (Building* building : this->getVecBuilding()){
        int portee = building->getPortee();
        Tile *bTile = building->getTile();
        float distance = bTile->distance(t->getTile());
        if (distance <= portee){
            building->upgradeTower(t);
        }
    }
}
```

Fonctions de parcours des vecteurs buildings et tours



5. Bresenham

Bien que l'architecture de notre map ne rende pas la fonction de Bresenham très pertinente, celle-ci étant plutôt dédiée à des tracés/parcours de courbe, nous avons tout de même pris le temps de l'implémenter. En effet, notre map étant très rectiligne, les monstres peuvent se contenter de bouger en ligne droite. Cela dit, notre fonction Bresenham recalcule bien la position du monstre en fonction de la marge d'erreur avec l'équation de la courbe souhaitée.

Pour choisir leur destination, nous regardons à quelles coordonnées se situe le monstre, il avance tant qu'il n'a pas atteint sa "destination". Une fois qu'il l'a atteint, il choisit le noeud suivant. Pour le moment, nous n'avons qu'un seul successeur pour nos nodes puisque notre map ne comportait qu'un seul chemin au moment de sa création. Il faudra bien évidemment utiliser Dijkstra pour déterminer quel chemin est le plus court. Mais cela nécessitait un remaniement de notre structure Node et nous avons préféré nous concentrer sur d'autres fonctionnalités.

```
if (deltaY1<=deltaX1){
    if(deltaX>=0){
        x=initialX;
        y=initialY;
        errorX=destinX;
    } else {
        x=destinX;
        y=destinY;
        errorX=initialX;
    }
    this->setX(x);
    this->setY(y);
    this->drawCat(this->getTexture(),x,y);
    for (int i=0; x<errorX;i++){
        if(pX<0){
            pX=pX+2*deltaY1;
        } else {
            if((deltaX<0 && deltaY<0) || (deltaX>0 && deltaY>0)){
                y=y+1;
            } else {
                y=y-1;
            }
            pX=pX+2*(deltaY1-deltaX1);
        }
        this->setX(x);
        this->setY(y);
        this->drawCat(this->getTexture(),x,y);
    }
}
```

Extrait de Bresenham dans move().



V) Difficultés rencontrées

Ce projet est assez conséquent, d'autant plus lorsqu'il s'agit d'un nouveau langage, il constituait un véritable challenge et nous a confronté à de nombreux obstacles.

1. Référence indéfinie

Lorsque nous avons voulu exécuter notre première version du code, après avoir corrigé les erreurs de compilation, nous nous sommes heurtées à un problème dont nous n'avons pas véritablement réussi à comprendre l'origine.

```
[ 30%] Linking CXX executable itd_eva
CMakeFiles/itd_eva.dir/src/Map.cpp.o : Dans la fonction « Map::readITD(char*) » :
Map.cpp:(.text+0x7f0) : référence indéfinie vers « Node::setX(float) »
Map.cpp:(.text+0x810) : référence indéfinie vers « Node::setY(float) »
Map.cpp:(.text+0x824) : référence indéfinie vers « Node::setType(int) »
Map.cpp:(.text+0x838) : référence indéfinie vers « Node::setIndex(int) »
Map.cpp:(.text+0x84c) : référence indéfinie vers « Node::setSuccessor(int) »
collect2: error: ld returned 1 exit status
CMakeFiles/itd_eva.dir/build.make:331 : la recette pour la cible « itd_eva » a échouée
make[2]: *** [itd_eva] Erreur 1
ebenhari@pccoplbl10-04:~/Documents/TowerDefense SANGARE BENHARIRA debug/TowerDefense SANGARE BENHARIRA-masters$
```

Erreur rencontrée dans la V1 du projet

Comme vous pouvez le voir sur cette capture d'écran, l'erreur survient dans notre fonction readITD. Celle-ci, située dans la classe Map (Map.hpp et Map.cpp), fait appel à une autre classe : Node, dans le but de créer un tableau de Node.

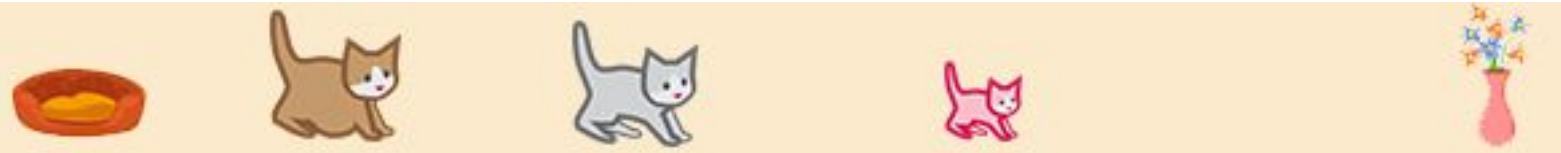
Cependant, bien que nous ayons inclus comme il se doit les fichiers et vérifié de multiples fois nos pointeurs et nos inclusions, nous n'avons pas réussi à passer outre cette erreur et c'est pourquoi nous avons développé une nouvelle version de notre code.

Les différents camarades auxquels nous avons fait appel ne sont pas parvenus à nous aider et certains ont également rencontrés le même problème avant d'abandonner la structure Node, faute de solution. Nous ne pouvions décemment pas recréer l'entièreté de notre structure Node si proches de la date des rendus. C'est pourquoi, à partir de ce moment là, nous avons décidé de créer une V2 à notre projet.

2. Gestion du fichier principal Game

Comme expliqué au préalable, lors de notre réflexion autour du sujet, nous avons décidé d'implémenter une classe Game pour gérer les différents aspects du gameplay à proprement parler de notre jeu. Cependant, cela s'est révélé plus compliqué que prévu.

En effet, créer une telle classe nous exposait à des risques de redondance et de double inclusion puisque nous avons besoin de faire appel à elle dans d'autres classes tandis que toutes nos classes étaient déjà incluses dans Game.hpp et Game.cpp.




Nous avons donc dû faire appel à une notion qui n'avait été que brièvement évoquée en cours; la pré déclaration. Celle-ci consiste à créer une variable "class Game;" dans les fichiers concernés pour que ceux-ci en connaissent les attributs sans qu'elle ne soit véritablement incluse dedans.

De plus, nous voulions que les entités (monstres, tours, buildings) puissent accéder aux autres entités présentes dans le jeu, grâce aux attributs vector de Game. Nous avons donc dû créer un pointeur sur Game dans nos entités, bien que celles-ci soient incluses dans Game lui-même. Cela compliquait les choses, et n'a pas été implémenté dans notre version simplifiée (pour le moment).

3. Adaptation aux nouveaux langages

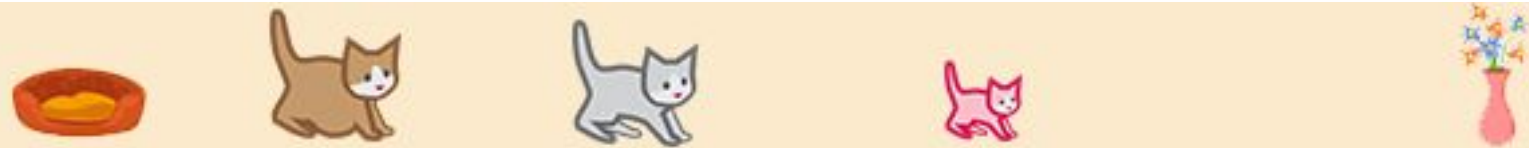
Nous avons également rencontré des difficultés liées directement à notre découverte d'un nouveau langage, le C++ et de nouvelles librairies.

La création d'un makefile a été notre premier obstacle.

▼ 7 ■■■ Makefile 

		@@ -4,11 +4,12 @@
4	4	# OUR MAKEFILE
5	5	
6	6	#compilateur
7		- CC = gcc
	7	+ #CC = gcc
	8	+ CXX = g++
8	9	#Variable Tower_defense
9	10	ITD = TowerDefense_SANGARE_BENHARIRA
10	11	#options de compilation
11		- CFLAGS = -Wall -O2 -g
	12	+ CPPFLAGS = -Wall -O2 -g
12	13	#options de l'édition de lien
13	14	LDFLAGS = -lSDL -lSDL_image -lGLU -lGL -lglut -lm
14	15	
		@@ -42,7 +43,7 @@ clean: #Remove all .o files
42	43	
43	44	BIGCLEAN: clean #Remove all "build artifacts" like .o files and the .exe
44	45	## ATTENTION ==> RISQUE DE TOUT EFFACER ?
45		- rm -rf \$(ITD)
	46	+ rm -f \$(ITD)
46	47	@echo "File .o and .exe are all removed"
47	48	

Capture d'écran des premières tentatives de MakeFile



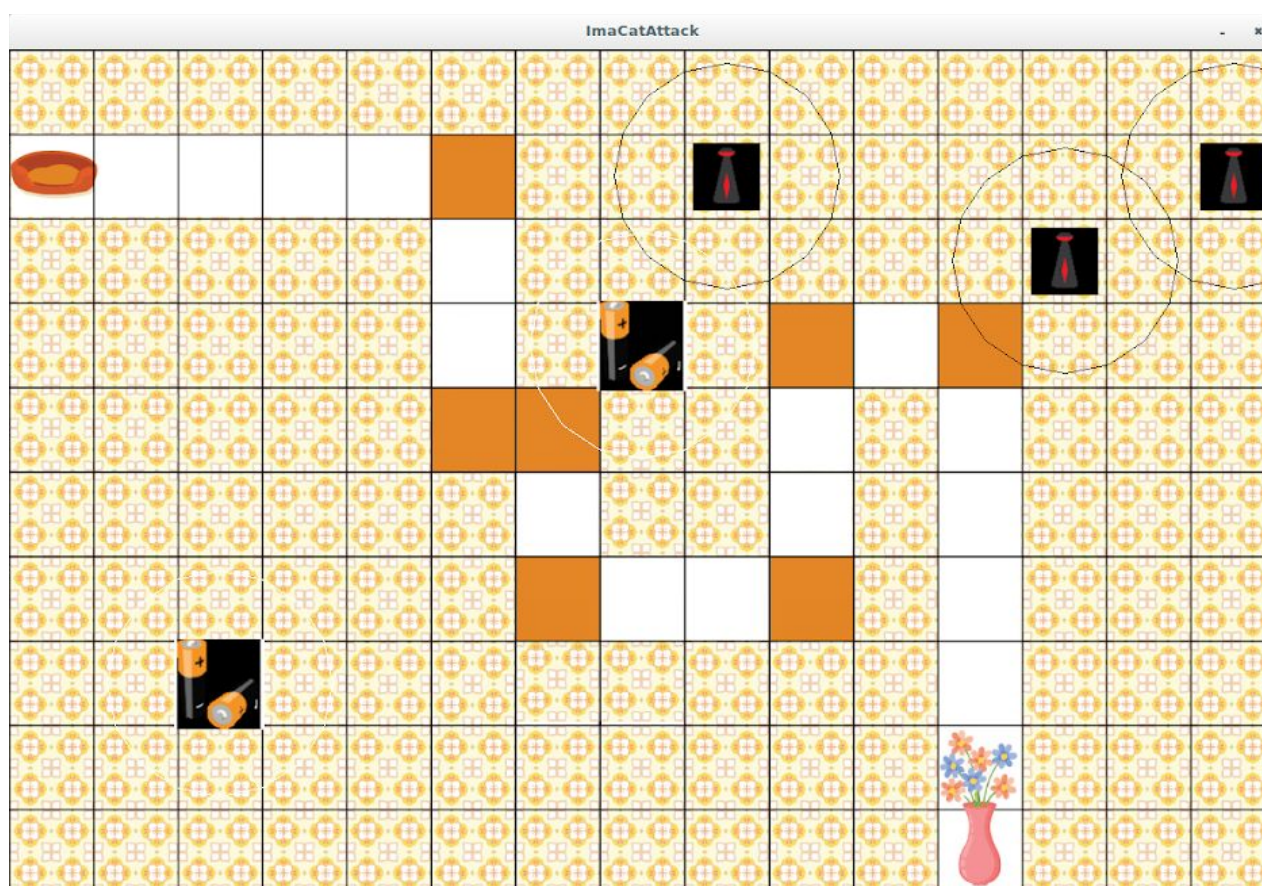
L'emploi du langage C++ nous poussait notamment à modifier les balises et les inclusions de bibliothèques. Cependant, nous avons trouvé un camarade disposé à nous expliquer la méthode **cmake**. Celle-ci semblant être utile pour l'an prochain nous avons profité de l'occasion pour nous y initier. C'est donc notre fichier CMakeLists qui crée finalement notre makefile et nous permet de compiler. C'est à ce moment également que l'on nous a conseillé d'utiliser "#pragma once" plutôt que les ifndef dans nos .hpp car il semblerait que cela diminue les risques de bug.

Nous avons aussi rencontré des difficultés concernant le côté interface du projet. En effet, les nuances entre SDL 1 et SDL 2 ne nous apparaissent pas clairement et nous peinions à prévoir les conflits. Il était difficile de savoir si les indications que nous trouvions sur internet étaient valables pour SDL1, SL2 ou les deux.

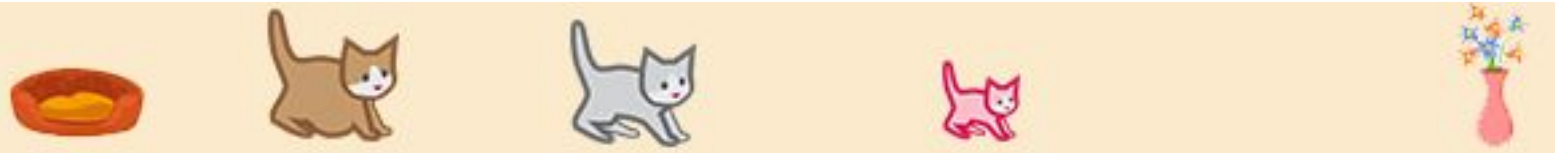
Enfin, étant des novices en manipulation d'OPEN-GL, il nous est arrivé de nous retrouver confrontées à des erreurs pouvant se résoudre en quelques lignes car nous ne comprenions pas que celles-ci manquaient.

Ce fut par exemple le cas lorsque que nous avons voulu gérer la transparence de nos sprites tours et buildings. Nous avons beau prendre en compte la valeur alpha, sans les lignes suivantes, cela ne pouvait pas marcher :

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```



Affichage de l'interface lors du problème de gestion de l'alpha



VI) Résultats

Légende	ALGORITHMIE		INFOGRAPHIE	
	Fonctions	Niveau d'implémentation	Fonctions	Niveau d'implémentation
Implémentée				
Non-implémentée				
Validée				
	Read ITD		Affiche map	
	Read PPM		Affiche entités	
	Vérif ITD/PPM (cohérence)		Réactualise entités (mouvement chat)	
	Gestion des vagues		Gestions d'événements	
	Move() + Bresenham		Buildable or not	
	Gestion de la cagnotte		Read PPM / Read ITD	
	Calcul de la distance		Création et suppression des entités	
	Building influe sur caractéristique tours			
	Création et destruction des entités			
	Attaques des tours et mort des monstres			
	Buildable or not			
	Nodes			
	Gestions d'événements			
	Utilisation de Dijkstra			

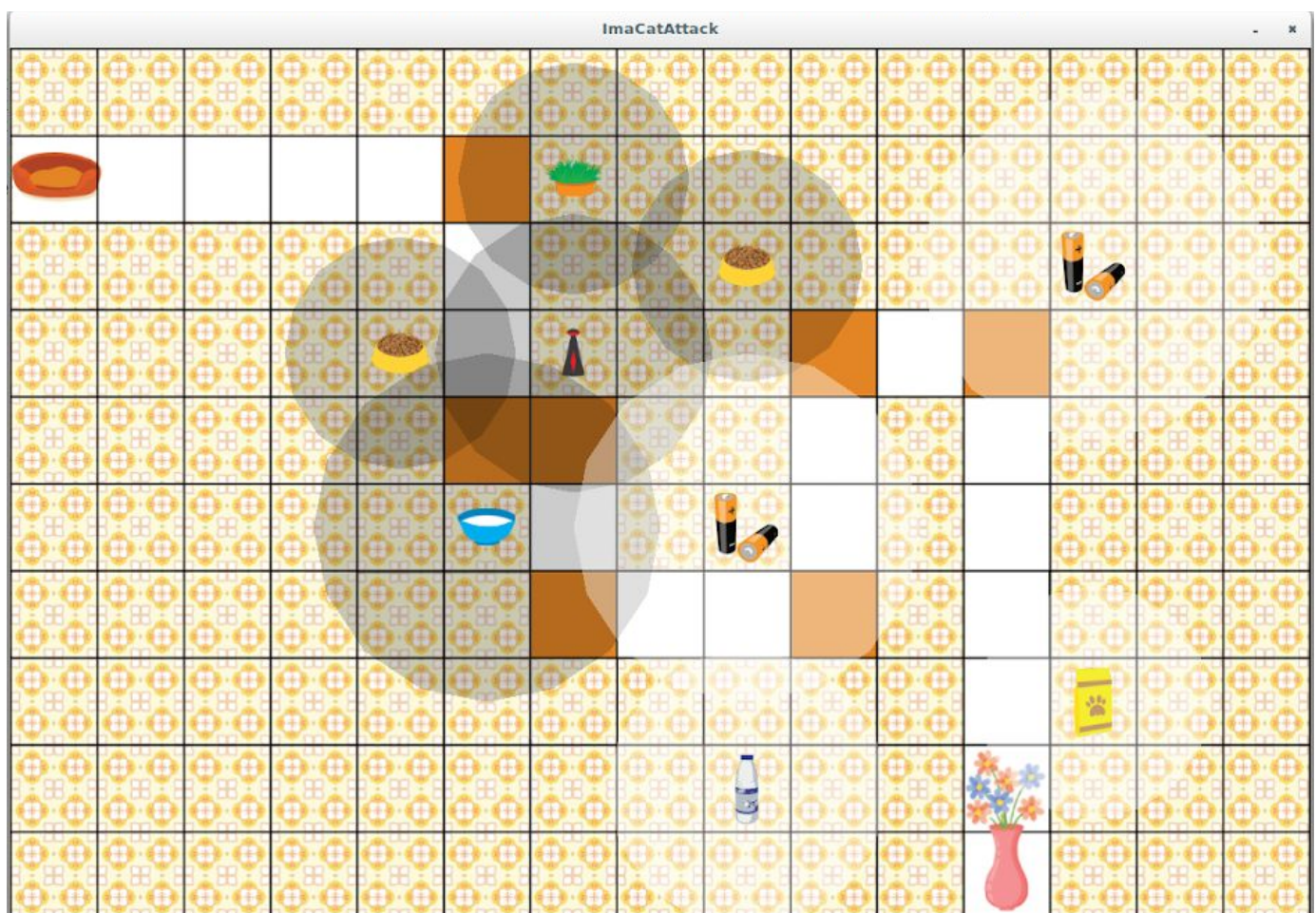
1. Côté algorithmie

Côté algorithmique, la fonction principale que nous n'avons pas implémentée est l'utilisation de l'algorithme de Dijkstra. En effet, n'ayant qu'un unique chemin, nous avons préféré ne pas perdre de temps sur une fonctionnalité qui n'allait pas nous servir dans l'immédiat. Bien évidemment, le principal problème de notre partie algorithmie est la présence de cette erreur que nous n'avons pas réussi à résoudre, malgré quelques pistes (qui pourraient aussi bien nous servir pour Dijkstra). A cause de cela, nous ne sommes pas aptes à tester le bon fonctionnement de l'ensemble de nos fonctionnalités. Cela dit, nous avons la confirmation (grâce à des printf entre autres) que la lecture du PPM et de l'ITD est correcte et fonctionne parfaitement.

2. Côté affichage

Notre version display du code nous a permis d'implémenter une bonne partie de nos fonctionnalités d'affichage notamment en appliquant des textures.

Comme vous pouvez le voir sur la capture d'écran ci-après, nous sommes à même d'afficher le fond de notre carte, ainsi que de créer des tours et des buildings. Ceux-ci, ne peuvent pas être construits sur la case d'entrée ou de sortie, ni sur le chemin. On ne peut créer qu'un bâtiment (tour ou building) par case. La transparence des png est bien prise en compte et nous avons même rajouté l'affichage d'une zone représentant la portée des tours et buildings.



Affichage avec placement de tours et de buildings sur la version affichage

Au-delà de la fenêtre à proprement parler, le terminal nous fournit également des feedbacks tout au long de notre interaction avec l'interface. Lorsque l'on est en mode construction de tour, il nous signale le type de tour qui correspond à la touche 1, 2, 3 ou 4 sur laquelle on a cliqué.

Il indique également lorsqu'une tour est créée et lorsque l'on a sélectionné une zone non constructible. Il en va de même pour les buildings.



VII) Pistes d'amélioration

1. Organisation

Nous avons consacré un certain temps au projet, cependant ne possédant pas Linux sur nos ordinateurs personnels, nous aurions dû prévoir plus de temps de travail sur les machines de l'université pour parvenir réellement à déboguer notre programme.

En effet, nous pensons être proches du résultat mais notre mauvaise gestion du projet ne nous permet pas d'atteindre l'objectif de présenter un jeu réellement fonctionnel dans les temps.

1.1. Constat

Rétrospectivement, commencer notre implémentation par les classes tours, monstres et buildings ne nous semble pas avoir été une si bonne idée. En effet, si cela s'adaptait sur le moment à nos compétences et à nos disponibilités respectives, nous avons mal évalué où se situait les difficultés. Procéder ainsi ne nous permettait pas véritablement de tester notre code et notre affichage dans la mesure où l'on ne peut afficher une tour sans afficher préalablement la carte.

C'est pourquoi si nous devons recommencer aujourd'hui, nous commencerions sans doute par gérer l'affichage de la carte et de ses chemins pour pouvoir nous rendre compte des problèmes qu'ils recèlent dès le début. **Partir du plus général au plus spécifique nous semble aujourd'hui une meilleure méthode.**

De manière générale, nous aurions à coeur de mieux découper nos tâches et de mieux prioriser nos implémentations. En clair, tester et compiler plus régulièrement pour se rendre compte des obstacles au jour le jour.

1.2. Objectifs

Comme nous l'avons évoqué notre accès à Linux était limité et nous a porté préjudice, cependant nous avons découvert qu'avec un éditeur de texte plus adapté (tel Visual Studio), nous devrions être à même de compiler sur nos propres PC. Cela devrait nous permettre d'avancer plus facilement sur des projets futurs.

De même, notre **débogage** pourrait se trouver **facilité** grâce à l'emploi d'un tel éditeur. Celui-ci est en effet à même de nous signaler les erreurs d'inclusions qui peuvent survenir par inadvertance. Il permet également de mettre en place un système de **breakpoints** qui nous permettrait de véritablement naviguer dans l'exécution de notre programme pour en comprendre les anomalies.

Enfin à l'avenir, si nous avons déjà une forme de gestion de projet qui nous a permis de nous répartir les tâches et de découper notre implémentation du code, nous aimerions aller plus loin en nous fixant notamment des **objectifs intermédiaires** (par exemple hebdomadaires) sur le modèle des sprints de la méthode agile. En effet, un tel fonctionnement nous permettrait de mieux suivre l'avancée d'un projet au long cours comme celui-ci de manière à pouvoir tirer le signal d'alarme plus tôt et ainsi pouvoir demander de l'aide de manière plus anticipée.



2. Fonctionnalités

L'amélioration la plus évidente consisterait à relier effectivement la partie algorithmie et la partie infographie. Ne pas avoir réussi cela constitue pour nous un vrai désappointement. Il nous faudrait soit trouver l'origine de l'erreur dans notre partie master, soit réimplémenter petit à petit nos fonctions dans une V3 pour s'assurer de leur fonctionnalités.

Réaliser la version simplifiée nous laisse penser que notre V1 pourrait être moins complexe architecturalement parlant, il serait donc intéressant que la "V3" soit une fusion des deux versions précédentes. Cela nous permettrait peut-être de limiter les risques d'erreurs d'inclusion notamment.

À partir de là nous pourrions tester notre gameplay et la façon dont nos différentes entités interagissent entre elles. Les tours ont-elle assez de puissance pour détruire les CatMonsters par exemple ?

De même la gestion de l'argent, cagnotte de départ et son actualisation au cours de la partie serait à tester pour s'assurer qu'elle est adaptée..

Des fonctionnalités simples pourraient alors être mises en place, notamment pour l'expérience utilisateur, comme l'affichage d'un menu "start" et d'un menu de fin.

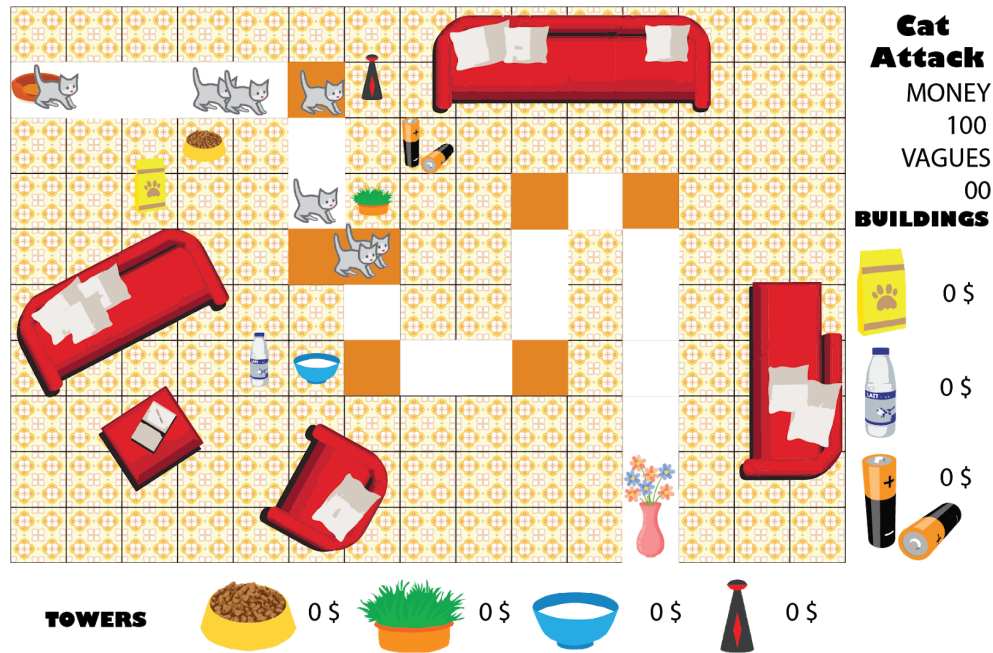
3. Expérience utilisateur

Au-delà de posséder un code fonctionnel, nous avons au cours du développement pensé à certaines fonctionnalités qui pourraient améliorer l'expérience utilisateur et mériteraient d'être développées.

Ainsi l'affichage d'une ou plusieurs barres de menu autour de la carte nous semble important pour que l'utilisateur puisse se rendre compte de l'avancée de la partie notamment, à travers le **compteur de vague** par exemple.

Une telle interface pourrait aussi lui fournir des informations sur **l'état de sa cagnotte**, ses gains et ses dépenses s'affichant en direct sur le compteur. Parallèlement à cela, afficher le prix des tours et des buildings nous semble être essentiel pour qu'il puisse ainsi gérer ses dépenses.

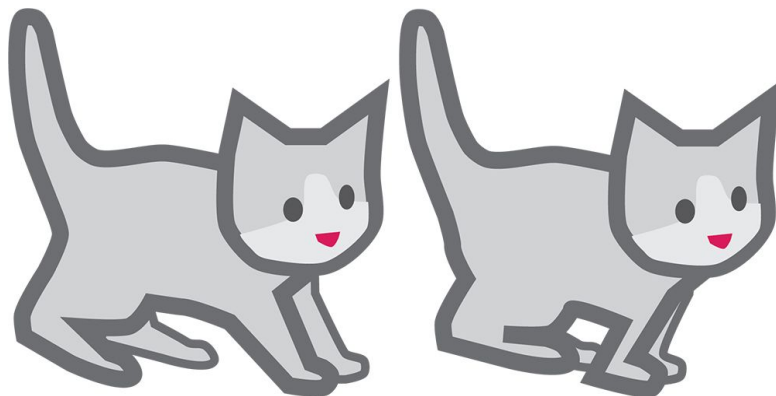
Nous pourrions aussi en plus du prix des tours, **afficher leurs autres caractéristiques** pour permettre au joueur de mettre en place des stratégies. Un **point d'aide** rappelant l'usage des touches pour construire tour et building serait également plus pratique pour l'utilisateur que de devoir se référer au readme ou au guide utilisateur (*voir Partie I.4.*).



Aperçu d'interface possible pour une version fonctionnelle

Également pour simplifier l'expérience utilisateur, notamment dans le choix du type de tour, nous avons prévu d'effectuer une **modification du curseur en fonction du type** de tour que celui-ci s'apprêtait à construire. Nous avons à cette intention créé des reproductions de nos images au format bmp. L'application de cette fonctionnalité serait la même que celle d'utiliser des textures. Notre image suivrait les déplacements de notre souris jusqu'à ce que la tour soit posée.

Enfin, pour rendre le jeu plus vivant, nous aurions aimé implémenter une lecture de sprite de manière à **animer nos chats** sur ce modèle :



Également, nous aurions aimé pouvoir mettre en place des **feed-backs** tels que des **sons** et une petite animation lors de la destruction des monstres et de la même manière un bruit de verre cassé si le joueur perd.



VIII) Retour sur expérience

1. Eva Benharira

Cette seconde expérience de projet informatique après le mini-gimp du second semestre me permet de voir ma progression sur cette année d'IMAC 1 qui constitue pour moi une véritable année de remise à niveau en programmation. Je me suis en effet familiarisée avec les différentes interfaces que nécessitent un tel projet (terminal, éditeur, git).

Créer les différentes classes (tower, building, CatMonsters, etc) m'a permis de réellement comprendre le fonctionnement du C++ et de la programmation objet. J'ai pu m'appuyer sur des notions vues en cours d'architecture logicielle pour créer les attributs de mes classes et sur mes tps de programmation et algorithmes pour la manipulation de la programmation orientée objet en général. Ce projet a été l'occasion de mettre en pratique ce que j'avais vu et d'aller plus loin. Des notions encore floues pour moi telles que les getters et les setters sont ainsi devenues beaucoup plus claires.

La réflexion que nous avons menée avec Laurelenn sur la mécanique de jeu et l'articulation de toutes nos fonctions a également été très intéressante pour moi pour comprendre en pratique l'intérêt de notion de classe abstraite ou d'héritage. De même, notre réflexion conjointe sur les fonctions readPPM et readITD a permis de mettre ma compréhension algorithmique à l'épreuve.

Enfin, si mes précédents projets m'avaient déjà permis de me familiariser avec l'emploi de git, ce projet m'a permis d'apprendre à me servir de git sous linux. J'ai ainsi appris à cloner mon répertoire ainsi qu'à réaliser des commits directement depuis mon terminal. Si cela n'est pas toujours facile, je sais maintenant que j'en suis capable.

Pour ce qui est de la partie infographie, je dois dire que créer les différents visuels a été très plaisant et que je suis très heureuse du scénario que nous avons su développer autour du sujet de Tower Defense. Je suis malheureusement déçue de ne pas pouvoir le voir "en action".

Ainsi même si le jeu que nous présentons n'est pas abouti, j'ai aimé travailler sur ce projet. Il m'a permis de constater que j'acquiers petit à petit des bases de programmation et d'ancrer les connaissances abordées ce semestre.

2. Laurelenn Sangaré

Ce serait mentir que de dire que je ne suis pas quelque peu déçue par le résultat de notre projet. Il est vrai que nous avons passé énormément de temps sur la partie algorithmique et qu'il est regrettable de ne pas voir ce projet plus abouti.

Eva et moi n'ayant pas la même expérience en développement informatique, nous avons jugé bon qu'elle commence à implémenter les fonctionnalités qu'elle avait déjà pu observer afin de fortifier ses bases, pendant que je m'occupais d'autres projets communs qui nécessitaient également un peu d'expérience et surtout de temps. Ayant réalisé le premier projet mini-gimp avec elle, je suis heureuse qu'en l'espace d'un semestre elle ait pu fortifier ses bases et d'avoir pu lui être utile.



Bien évidemment, j'ai beaucoup appris également. Il s'agissait uniquement de mon deuxième projet complet en programmation (le mini-gimp étant mon premier), et j'ai pu découvrir le C++, que je sais être très utilisé, ainsi que l'OpenGL. J'ai pu aussi découvrir l'existence de la structure "vecteurs", fortifier mes bases en pointeurs, utiliser les nodes dans un projet concret, etc.

Cela dit j'ai la fierté de pouvoir dire que nous avons su retomber sur nos pattes (*un peu comme les chats*) et avons pu développer une V2 avec interface graphique en moins de deux jours. Cette expérience m'a cependant beaucoup appris sur la gestion d'un projet de cette ampleur : il faut aller du plus général et du plus simple au plus spécifique et complexe, et savoir se recentrer lorsque l'on commence à s'éparpiller.



Conclusion

Bien que nous n'ayons pas effectivement réussi à obtenir un jeu fonctionnel, cette expérience nous a permis de réfléchir pour la première fois au fonctionnement d'un jeu vidéo. Nous avons algorithmiquement écrit toutes les fonctions demandées et donc réfléchi à leur implémentation.

Si la phase de debug s'est avérée plus difficile que prévu, celle-ci nous a beaucoup appris sur nous et nos méthodes de travail. Notre réflexion s'est peut-être en effet tournée trop vite vers la structure et son implémentation sans réfléchir à un découpage de fonctions effectivement testables.

Cependant, coder ce projet en C++ nous a permis d'apprendre pour l'une à utiliser la programmation orientée objet et de renforcer ses connaissances pour l'autre.

Ce projet a été une source de nombreuses expérimentations qui pourront nous resservir par la suite : la création et l'utilisation d'un cmake, l'emploi de git sous linux et la réflexion sur des algorithmes classiques comme Bresenham.

Ce sujet très complet, nous a permis de développer notre capacité à nous adapter à une situation problématique et de réagir en conséquence. Créer deux versions du projet n'était pas demandé mais c'est notre manière de certifier que nous avons bel et bien acquis des connaissances et que nous savons les appliquer.

Cette expérience nous a beaucoup appris en développement, mais aussi en gestion de projet, ce qui fait partie des compétences qu'un ingénieur devrait avoir. C'est pourquoi, malgré le résultat final, nous savons que nous en ressortons conscientes des erreurs que nous avons faites et que nous ne répéterons plus.

