Lilian FRANCHI
Pierre WARGNIER

Licence 3 Informatique
2014-2015

Defense de Tour

Encadrant : Julien Bernard

Table des matières

REMERCIEMENTS

Nous tenons, tout d’abord, à remercier M. Julien Bernard pour avoir été notre encadrant. Il a su nous soutenir et nous guider tout au long de ce projet. Il nous a montré nos erreurs et nous a encouragés à trouver les solutions par nous-mêmes, tout en se montrant disponible si nous étions en difficulté. Nous avons ainsi bénéficié d'une grande liberté dans nos choix de développement.

Nous souhaitons également remercier L'Université de Franche-Comté, pour avoir mis à notre disposition les ressources nécessaires au bon déroulement de notre projet.

INTRODUCTION

Durant la troisième année de Licence Informatique, dans le but de préparer les élèves à des projets conséquents en groupe, l'Université de Franche-Comté propose aux étudiants de réaliser un important projet tutoré, sur environ 3 mois, qui sera conclu par une présentation du travail réalisé, devant un jury de professionnels et d'étudiants.

Les projets sont proposés par les professeurs, et les binômes doivent faire une liste des projets qui les intéressent, ceux-ci seront ensuite attribués.

Pour nous, ce fut l'occasion de travailler avec une technologie que nous ne connaissions pas ou peu. C'est dans cette idée que nous nous sommes tournés vers les sujets des Jeux Vidéo proposés.

Le jeu vidéo, d'un point de vue programmation, rassemble divers éléments (images, sons, textes, ...) qui sont utilisés par un programme qui gère également les intéractions du joueur. Ce programme peut-être écrit dans divers langages (Java, C++, Python, …).

Pour un développeur, il faut une bonne maîtrise dans le langage choisi pour réaliser un jeu fini et complet. Mais le jeu vidéo, de par la diversité qui le compose, est aussi un bon moyen pour découvrir un langage de programmation, de connaître ses atouts, ses limites, et d'apprendre à être capable de s'en servir dans des situations inédites.

Nous avons donc demandé à faire un sujet sur un jeu vidéo et il nous a été attribué le sujet de Défense de Tour, à implémenter en C++ avec la bibliothèque SFML.

Nous vous présenterons dans un premier temps ce qu'est un jeu de défense de tour, et quels choix nous avons faits pour le nôtre. Dans un second temps, nous vous expliquerons comment nous avons réalisé nos choix. Pour conclure, nous parlerons des objectifs accomplis, de ceux non-réalisés, puis des améliorations possibles.

PRESENTATION

Défense de Tour

Un jeu de défense de tour, couramment appelé Tower Defense (terme que nous utiliserons par la suite), est un type de jeu vidéo où l'objectif est de défendre une zone contre des vagues successives d'ennemis, se déplaçant suivant un itinéraire ou non, en construisant et en améliorant progressivement des tours défensives.

On date l'arrivée du Tower Defense, comme un genre de jeu, au début des années 90. Même si on connaît l'existence de tower defense avant 1998, c'est à partir de cette date que les bases telles que nous les connaissons sont posées par le jeu de stratégie StraCraft, développé par *Blizzard*. Relayé ensuite dans des jeux populaires, tel que Warcraft 3, le Tower Defense prend alors l'essor que nous lui connaissons. De plus, avec la technologie flash et l'expansion d'internet, le Tower Defense se démocratise sur le net, grâce à sa facilité d'accès, d'utilisation, et sa gratuité.

Quelque soit le Tower Defense choisi, on retrouve toujours des éléments de gameplay de base, avec des modifications propres à chaque jeu. Dans ces éléments de base, on retrouve:

- Un Lanceur, couramment appelé «Tour», dont le but est d'empêcher les ennemis de parcourir leur itinéraire. Les Tours se caractérisent par:
    - Leur coût,
    - Leur vitesse d'attaque,
    - Leur «DPS» (Dégâts par Seconde),
    - Leur portée;
    - Leur type d'attaque (Terrestre, Aérien,...),
    - Certains malus qu'elle inflige à l'ennemi (ralentissement,...),
    - Et leur immobilité (bien que cela reste dépendant du développeur)

- Un Coureur, couramment appelé «Ennemi», dont le but est de parcourir un itinéraire. Les Ennemis se caractérisent par:
    - Leur résistance,
    - Leur vitesse,
    - Leur capacité à détruire les Tours ou non,
    - Leur résistance aux malus infligés par les Tours,
    - L'obligation de suivre le chemin défini par le joueur ou le niveau

Warcraft 3

StarCraft 2

Notre Tower Defense

Le but de notre Tower Defense est de générer, à partir d'un fichier texte simple, un niveau, avec un point de départ, d'arrivée, et un chemin prédéfini. Les ennemis devront pouvoir le parcourir, plus ou moins vite, selon l'ennemi. Les tours ne pourront pas être posées sur l'itinéraire des ennemis (incluant le départ et l'arrivée), et tireront des boules plus ou moins vite, et qui feront plus ou moins de dégâts selon la tour. Des élèments de gameplay, tel que l'argent ou la vie, seront également implémentés.

Le C++ est un langage très utilisé dans le développement de jeu vidéo. Il est également connu pour avoir beaucoup de bibliothèque en complément. De plus, notre connaissance en C++ est rudimentaire, et nous voulons saisir la chance qui nous est offerte de parfaire notre connaissance. Nous espérons que la réalisation du Tower Defense nous permettra de voir le potentiel de ce langage.

Nous utiliserons la Bibliothèque SFML pour gérer la partie graphique, car celle-ci est écrite en C++ et possède presque tous les modules que nous voulons (image, événement,...); nous n'avons ainsi qu'un seul SDK pour toute ces fonctionnalités. Elle est également très bien expliquée grâce aux nombreux tutoriels présents sur le net. Nous utiliserons également la Bibliothèque Boost pour la gestion des fichiers.

Pour communiquer, nous avons décidé d'utiliser un logiciel de gestion de version, Git, car nous connaissions tous deux ce logiciel et cela nous permettra d'optimiser les temps de travail de chacun.

Pour le Design, nous avons décidé de faire quelque chose de minimaliste, avec essentiellement des images trouvées sur internet. Le jeu ne comportera pas de son. L'IDE (environnement de développement intégré) utilisé sera CodeBlocks sous Windows.

IMPLEMENTATION

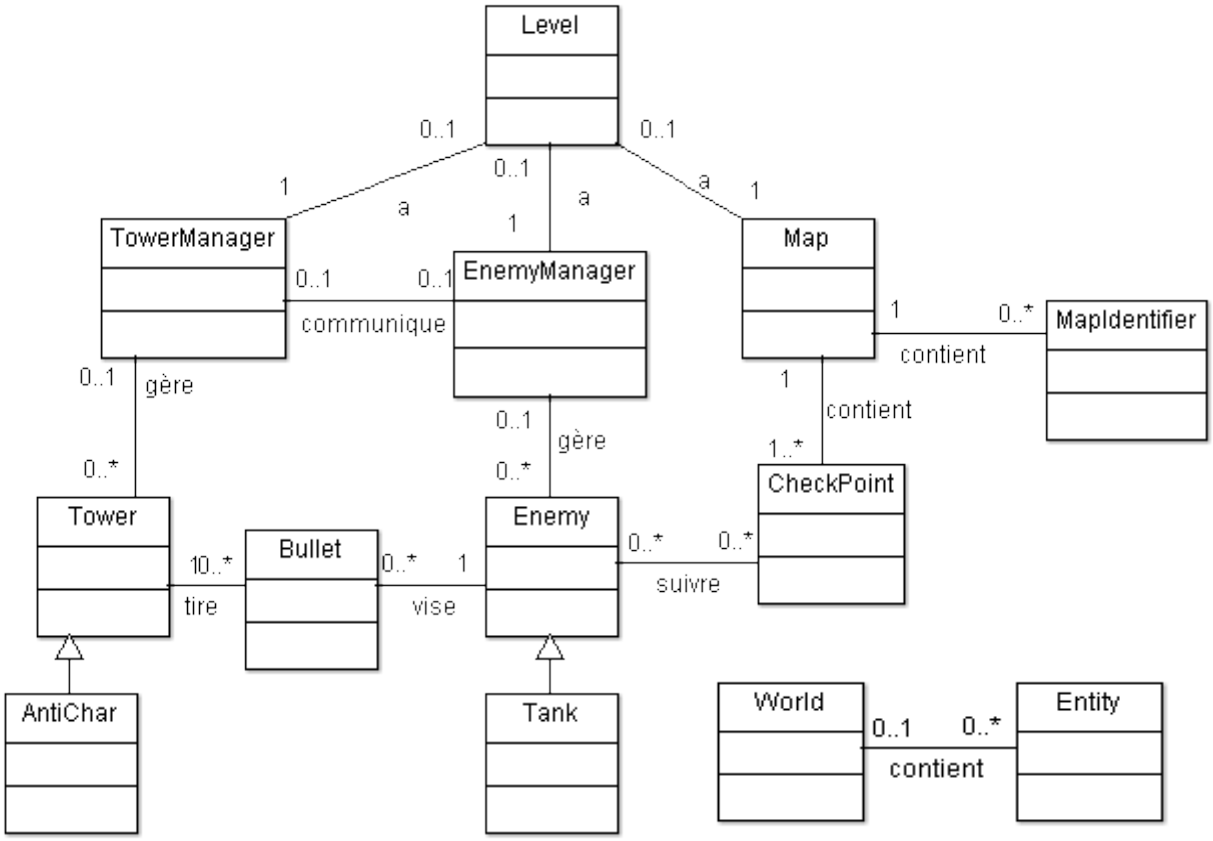
En C++, il est d'usage de séparer prototype (déclaration) et implémentation (définition) de classe dans deux fichiers:

- La déclaration se fait dans un fichier d'en-tête «header»:
  - sans extension dans le standard,
  - .h comme en C,
  - .hh,
  - .hpp

- La définition de la classe se fera dans un fichier source ,d'extension également variable, en général:
  - .C,
  - .cc,
  - .cpp,
  - .cxx

Lorsque nous parlerons de classe, nous sous-entendrons l'existence des deux fichiers.

Modélisation



La classe Entity est la classe dont hérite les principales classes de notre tower defense. Ainsi, Level, TowerManger, EnemyManager, Bullet, Enemy, Tower, Map, héritent d'Entity.

La classe World est utilisée dans le main, une fois un attribut world créée, on l'utilise pour gérer les Entity du tower defense. (carte, niveau, gestionnaire des ennemis, gestionnaires des tours, ...)

Chargement de la Carte

Comme dit précédemment, nous avons décidé que nos niveaux seraient générés à partir d'un fichier texte de la forme:

```
#D########
#.#2....3#
#0.1####.#
########.#
########A#
```

#: endroit où l'on peut poser une tourelle et où les ennemis ne peuvent pas aller

. : chemin des ennemis

D : départ des ennemis

A: arrivée des ennemis

0..9: Chiffre correspondant à un CheckPoint

Pour ce faire, nous avons une classe Map, qui prend en paramètre le chemin d'accès à ce fichier, ainsi que la taille de la fenêtre. Ensuite, après avoir ouvert le fichier, celle-ci le parcourt à l'aide du code suivant :

```
if (levelTxt.is_open())
{
  unsigned int i=0;
  int j=0;
  while(getline(levelTxt,line))
  {
    m_level.push_back(std::vector<MapIdentifier>());
    for(i=0; i<line.length(); i++){
```

A chaque élément rencontré, Map met l'identifiant sous lequel est connu l'élément dans un tableau d'identifiant qui, à terme, représentera le niveau. Ainsi, le fichier texte est lu une seule fois.

Affichage de la Carte

L'affichage de la carte est réalisé dans la classe Map à l'aide de la fonction render.

```
for(unsigned int i=0; i<m_level.size(); ++i){
  std::vector<MapIdentifier> line = m_level[i];
  for(unsigned int j=0; j<line.size(); ++j){
    switch(line[j]){
    case MapIdentifier::FIELD:
      texture = ImageHandler::getTexture(SpriteList::FIELD);
      break;
```

Celle-ci parcourt le tableau contenant le niveau, ligne par ligne, puis case par case, puis, pour chaque élément du tableau, la fonction fait appel à la classe ImageHandler, dont le rôle est de charger les textures des éléments.

```
sprite.setTexture(texture);
sprite.setPosition(j*m_tileWidth, i*m_tileHeight);
window.draw(sprite);
```

Ainsi, la fonction render de map peut créer le sprite et l'afficher comme indiqué ci-dessus. Pour l'affichage des ennemis et des tours, le procédé reste le même, sauf pour la position qui est cette fois déterminée par la position de l'élément dans le jeu.

Ainsi, on peut afficher :

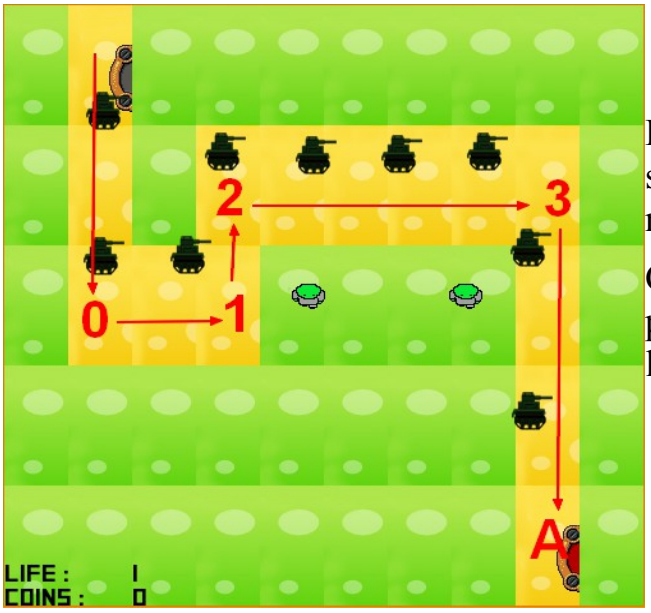
Ennemi

A ce stade, nous sommes capables d'afficher les éléments du jeu, mais il faut maintenant leur donner vie.

I Le déplacement :

Précédemment, nous avons dit que le fichier texte qui crée le niveau possédait des points de passage, numérotés de 0 à 9. Lors de la création d'un niveau, on place les points de passages de façon à ce que l'ennemi fasse le chemin voulu.

De cette façon, on obtient :



De part le format de nos niveaux, nous sommes limités à 10 point de passage par niveau (numéroté de 0 à 9) .

Cela limite donc la complexité que nous pouvons donner à nos niveaux, ainsi que leur taille.

Concrètement, lorsque l'on veut faire avancer un ennemi, on regarde le dernier point de passage utilisé par l'ennemi, ainsi que la position de l'ennemi. Si l'ennemi est dans les environs du point de passage (une distance proche, quelque pixel), alors on passe au point de passage suivant.

Le tir des tours est fait de la même façon, sauf que les points de passage sont remplacés par les cibles.

II Nombre d'ennemis et fréquence de sortie :

Nous avons indiqué le nombre d'ennemis directement dans le code, ainsi, chaque niveau a 10 ennemis, crées par EnemyManager. De même pour la fréquence de sortie de la base, qui est de 1.5 secondes.

```
sf::Time elapsed = clock.restart();
world.update(elapsed.asSeconds());
```

Pour ce faire, on utilise une classe disponible grâce à SFML : Clock.

Ainsi on peut mesurer le temps écoulé, et grâce à un compteur dans EnemyManager, on peut décider de rajouter un ennemi seulement toutes les 1.5 secondes minimun.

Tour

Maintenant que les ennemis bougent correctement, nous allons parler des tours.

I Le positionnement :

Les tours sont positionnées par le joueur. Elles ne peuvent ni se chevaucher, ni tirer sur un ennemi hors de portée.

Le joueur pose une tour uniquement s' il a suffisamment d'argent, et si la position à laquelle il veut la mettre est correcte.

```
if(line[(int)(x/50)]==MapIdentifier::FIELD
   && line[(int)((x+ImageHandler::getTexture(SpriteList::TOWER).getSize().x)/50)]==MapIdentifier::FIELD
   && line_bot[(int)(x/50)]==MapIdentifier::FIELD
   && line_bot[(int)((x+ImageHandler::getTexture(SpriteList::TOWER).getSize().x)/50)]==MapIdentifier::FIELD
   && !alreadyTower(x,y)){
  allTower.push_back(new Tower(1, 50, x, y, m_emanager));
  return true;
}
```

Une position Correct est une position comportant le symbole # dans le fichier texte.

Dans le code ci-dessus, on teste que la position à laquelle le joueur veut poser la tour est bien un # (FIELD) et que la position est libre, si c'est le cas, alors on pose la tour.

II Le tire sur les ennemis :

Comme dit précédemment, les tours ne peuvent tirer sur les ennemis que s' ils sont à portée. Pour ce faire :

```
for(Enemy* e : m_emanager->getAllEnemies()){
  if((e->GetPosX()-m_posX/50.0 <= 2.1 && e->GetPosX()-m_posX/50.0 >= -2.1)
     && (e->GetPosY()-m_posY/100.0 <= 2.1 && e->GetPosY()-m_posY/100.0 >= -2.1)
     && m_dt_cumulated > 1){
    m_bullets.push_back(new Bullet(m_posX+ImageHandler::getTexture(SpriteList::TOWER)
    m_dt_cumulated = 0;
  }
}
```

On parcourt tous les ennemis présents dans le niveau et on teste si l'ennemi en question est à portée, si c'est le cas, on crée un boulet pour lui tirer dessus, sinon on passe à l'ennemi suivant.

L'ennemi choisi sera le premier de la pile à être à portée, et la tour lui tirera dessus jusqu'à ce qu'il soit mort ou qu'il soit hors de portée. Puis, elle passera au suivant.

Le déplacement du boulet jusqu'à l'ennemi est fait de la même façon que le déplacement de l'ennemi au checkpoint, sauf que la position de l'ennemi est actualisée à chaque mise à jour.

Niveau

Maintenant que nous avons une carte, des ennemis qui se déplacent, des tours qui tirent et qui sont posées par le joueur, nous allons pouvoir faire un Niveau.

Pour faire un niveau, ils nous faut déjà assembler tous les éléments précédents. Il nous faut également rajouter quelques mécaniques pour le rendre intéressant:

L'Argent:

Déjà évoqué comme condition pour poser une tour, il faut ajouter que l'argent est gagné à chaque mort d'ennemis, à hauteur de 10. Une tour coûtant 50.

Pour ce faire, nous avons rajouté un compteur dans la classe level qui est mis à jour à chaque mort d'ennemis et achat de tours.

Les Points de vie:

Les points de vie représentent le nombre d'ennemis qui peuvent atteindre le point d'arrivée avant que le niveau ne soit considéré comme perdu. Ceux-ci sont fixés directement dans le code et décrémentés dès qu'un ennemi atteint l'arrivée. Lorsque ceux-ci arrivent à 0, le joueur a perdu.

Victoire

Pour gagner, un joueur doit avoir éliminé la totalité des ennemis. Une fois cela accompli, il change de niveau.

Le changement de niveau est fait directement dans la boucle de jeu.

```
if(level.goodEnd() && !level.badEnd()){
  mapLevel.changeLevel("res/maps/level2.txt");
  level.changeLevel(1);
  eMan.nextLevel(10, &mapLevel);
}
```

Si le joueur a gagné, alors on change de niveau, c'est-à-dire qu'on charge la nouvelle carte, que l'on remet à zéro la position des tours, que l'on fixe le nombre de points vie à la valeur donnée, que l'on remet à 100 l'argent possédé, et que l'on recharge les ennemis.

Pause:

A tout moment en cours de partie, le joueur peut appuyer sur P pour mettre le jeu en pause.

```
case sf::Keyboard::P:
  level.pause();
  break;
```

Lorsque le joueur appui sut P, le level transmet l'information au TowerManager et à l'EnnemyManager. Les deux classes passent leur variable pause à vrai, et arrêtent d'actualiser la position de leur entité tant que cette variable est vrai. Ainsi, les objets tel que les ennemis et les boulets sont fixes.

BILAN

Nous avons réussi à faire un jeu fonctionnel. C'est-à-dire que nous avons :

- Des ennemis se déplaçant et donnant un bonus en mourant (argent).
- Des tours que l'on pose à la souris, et coûtant de l'argent.
- des tours qui tire des boulets sur les ennemis pour empêcher leur progressions.
- La gestion de la portée des tours.
- Une gestion des points de vie du joueur, qu'il perd à chaque fois qu'un ennemi atteint sont objectif.
- Une gestion de la victoire du joueur, lorsque tout les ennemis sont détruit.
- Un changement automatique de niveau en cas de victoire.
- La possibilité de recommencer le niveau en cours en cas de défaite.
- La possibilité de mettre le jeu en pause à tout moment.
- Une création de carte à partir d'un fichier texte (pour modifier un niveau existant, il suffit de modifier le fichier texte qui l'engendre)

Si nous poursuivons le développement, voici quelques améliorations que ne aimerions faire :

- Graphique,
- rajouter des types d'ennemi, de tour, et des niveaux,
- rajouter l'action de supprimer une tour,
- rajouter l'action d'améliorer une tour,
- rajouter la possibilité de faire variée la vitesse de l'ennemi,
- changer la fréquence d'attaque des tours,
- créer un menu
- créer des niveaux de difficulté (facile, moyen, difficile)
- Dans le cas où les chemins sont multiples, faire en sorte que tout les ennemis ne suivent pas tous le même point de passage.

Venant tous deux de parcours différent (de L2 et de BTS), nous avons pu bénéficier de la diversité nos connaissances, ce qui nous a permis d'apprendre beaucoup sur ce projet. Nous avons très vite réussi à nous coordonner (notamment grâce à Git), ce qui nous a permis de nous plonger directement dans la partie la plus intéressante de notre projet : L'apprentissage du C++.

Aujourd'hui, nous avons la sensation d'avoir compris ce langage au mieux que nous le permettais ce projet, et serions à même de mener d'autre projet dans ce langage de manière beaucoup plus efficace. Nous avons appris à connaître l'essentiel des fonctions offertes par la bibliothèque SFML et avons appris à résoudre les problèmes courant propre à ce langage.

Cette expérience nous a apporté énormément de connaissance diverse, notamment sur l'histoire du jeu vidéo, et sur la gestion de l'humain. Car bien que nous n'ayons pas eu de problème d'organisation, celle-ci à du ce faire autour de la vie de chacun.

Étant donné le plaisir que nous avons pris à développer ce projet, nous continuerons sûrement, chacun de notre coté, ou ensemble, à développer de petites fonctionnalités pour rendre notre jeu plus jouable.

ANNEXES

Carnet de Bord :

Ce carnet de bord est extrait d'un cahier que nous avons remplit au fur et à mesure de l'avancement de notre projet.

Semaine 1 : Du 10/11 au 16/11

Objectif :

- Avoir une version 0 du jeu pour le 09/12.

Travail réalisé :

Après réflexion, nous avons décidé d'utiliser le même IDE sous Windows. Dans la mesure où l'application doit être implémentée en C++ utilisant la bibliothèque SFML, nous avons décidé d'utiliser CodeBlock, en raison des nombreux tutoriels présent sur Internet.

Étant tous les deux présent à DPS (Dead Pixel Society), nous avons déjà travaillé avec SFMl et donc nous avons décidé d'inclure la bibliothèque Boost pour la lecture de fichiers et libtmx pour pouvoir profiter du pack de démarrage offert par DPS.

Problèmes Rencontrés :

- L'installation de boost à pris plus de temps que prévu à cause de notre inexpérience.
- Quelques problèmes sont apparus avec SFML, nous les avons corrigés avec une désinstallation/réinstallation de CodeBlock.
- Des difficultés à se transmettre les données du projet.

Semaine 2 : Du 17/11 au 23/11

Objectifs :

- Créer un emplacement de travail sur Github.
- Créer et hiérarchiser notre répertoire de travail ;
- Utiliser libtmx.
- Se renseigner sur les tower defense existants.

Travail réalisé :

Le répertoire Mialus/towerDefense est crée sur Github.

TowerDefense

|->help->Contient les projets de L2

|->include

| |->towerdefense->Contient les .h du projet

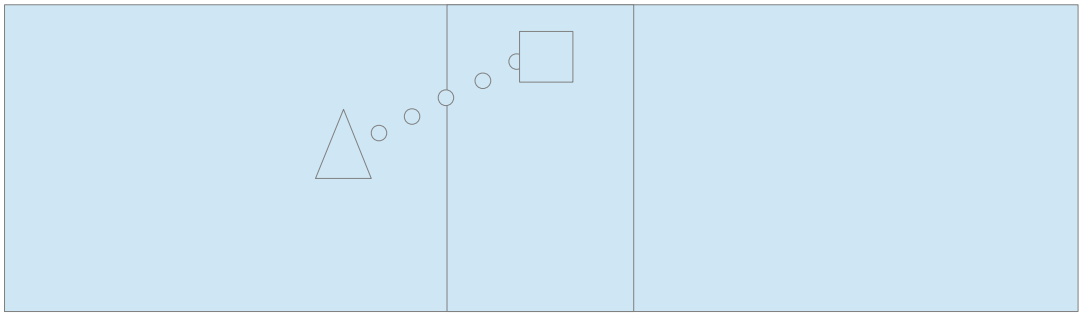
|-> res ->Contient les ressources

|->img->contient les textures

|->maps->contient les fichiers d'où sont générés les niveaux

|->src->répertoire qui contient tous les .cpp du projet

Suite à nos recherches, nous avons décidé que notre version 0 serait avec un niveau en ligne droite, un ennemi, une tourelle, un point d'arrivée et un point de départ de la forme :



Problèmes Rencontrés :

- Familiarisation avec les commandes git.
- Créer un répertoire qui puisse évoluer.
- Suite à de nombreux problèmes avec libtmx, nous avons décidé de ne pas l'utiliser.

Autres :

Format du fichier du niveau :

```
######
######
D...A
######
######
```

#: endroit où l'on peut poser une tourelle

. : chemin des ennemis

D : départ des ennemis

A : arrivée des ennemis

Semaine 3 : Du 24/11 au 30/11

Objectif :

- Créer les Entités Enemy, maps, tower, et imageHandler et les afficher.

Travail réalisé :

Les Entités Enemy sont créés, l'affichage n'est pas fonctionnel. L'affichage de maps est presque fonctionnel, tower et imageHandler sont en développement.

Problèmes Rencontrés :

- Découverte des fonctions graphiques de SFML.
- Le chemin vers les includes n'est pas pris en compte selon la machine sur laquelle l'application est située, la définition semble être indiquer correctement dans le projet.
- Problème avec la classe Ressource, ce qui a forcé l'utilisation d'une solution alternative.

Semaine 4 : Du 01/12 au 07/12

Objectifs :

- Terminer le travail de la semaine précédente.
- Créer les fichiers de gestions des éléments du Jeu.
- Avoir la version 0.

Travail réalisé :

La map du premier niveau est opérationnelle. Le format de fichier du niveau est défini ,comme indiqué précédemment, et interprétable par la classe map. Affichage de l'ennemi et de la tour , création de la classe gérant l'attaque de la tour, Création des textures des ennemis et des tours. Création de la fonction permettant le déplacement, modification de la classe enemy pour permettre le déplacement.

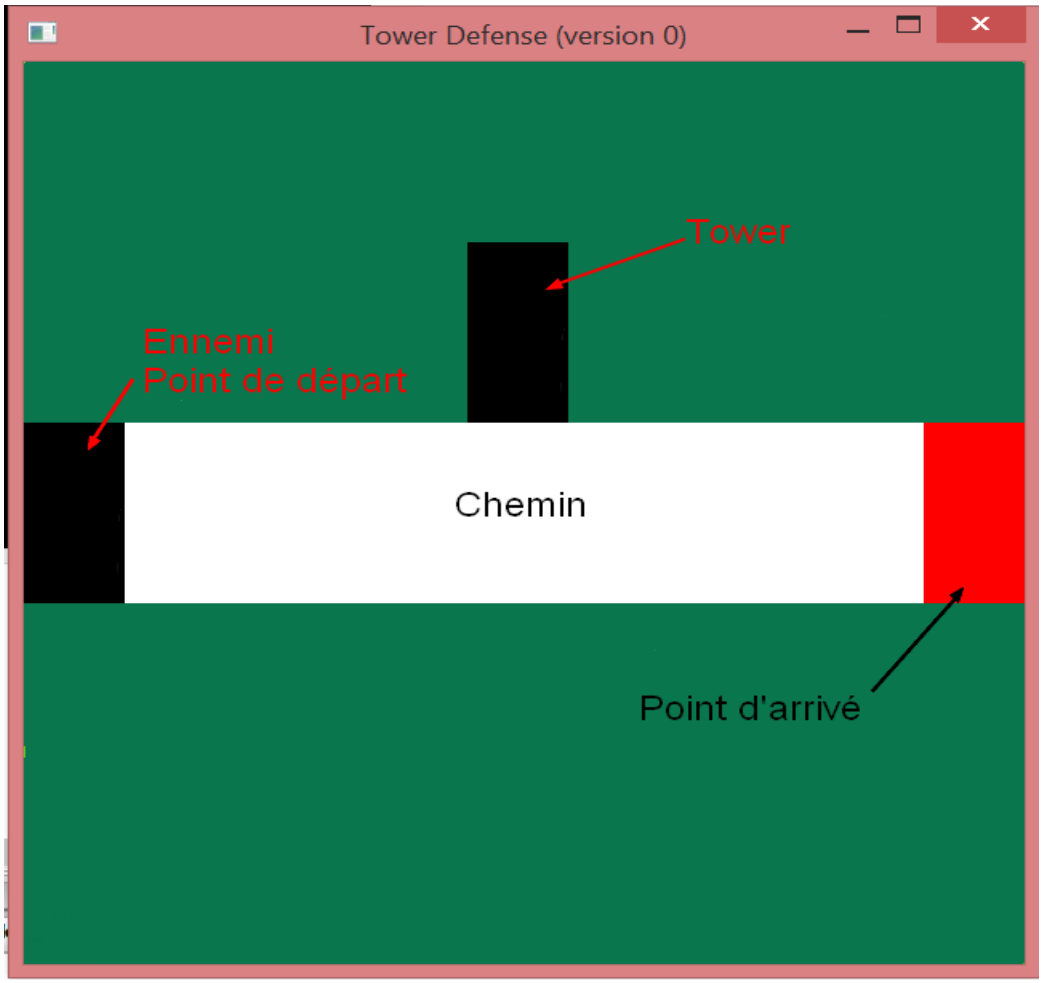
Problèmes Rencontrés :

- Utilisation de vectors pour les textures qui posent problème à l'affichage.
- Les ennemis et les tours sont de même couleur.
- Les ennemis ne se déplacent pas (problème lié à l'affichage des entités).
- Version 0 toujours pas opérationnelle.

Autres :

Le main est actuellement définit dans la classe Map. Les variables définies en int dans Tower et Enemy sont à mettre en float.

État du jeu :


Tower Defense (version 0)
Tower
Ennemi
Point de départ
Chemin
Point d'arrivé

Semaine 5 : Du 08/12 au 14/12 Et Semaine 6 : Du 15/12 au 21/12

Objectif :

- Correction et réimplémentation de certaines fonctions.

Travail réalisé :

La map du premier niveau est opérationnelle. Le format de fichier du niveau est défini ,comme indiqué précédemment, et interprétable par la classe map. Affichage de L'ennemi et de la tour , création de la classe gérant l'attaque de la tour, Création des textures des ennemis et des tours. Création de la fonction permettant le déplacement, modification de la classe Enemy pour permettre le déplacement.

Problèmes Rencontrés :

- Utilisation de vectors pour les textures qui posent problème à l'affichage.
- Les ennemis et les tours sont de même couleur.
- Les ennemis ne se déplacent pas (problème lié à l'affichage des entités).

Semaine 7 : Du 22/12 au 28/12

Objectif :

- Création de La Boucle de Jeu

Travail réalisé :

Nous avions déjà une boucle de Jeu « par défaut » qui nous permettait de tester nos implémentations. Nous avons donc implémenté une boucle de jeu plus complète afin qu'elle puisse prendre en compte toutes les intéractions que le joueur peut faire, ainsi que l'affichage et l'actualisation de tous les éléments du Jeu.

Problèmes Rencontrés :

- Le joueur ne peut plus poser de tour.
- Les ennemis ne se déplacent toujours pas.
- Des événements ne sont pas toujours pris en compte.

Semaine 8 : Du 05/01 au 11/01

Objectifs :

- faire bouger les ennemis
- changer les graphismes
- Corriger les problèmes des événements

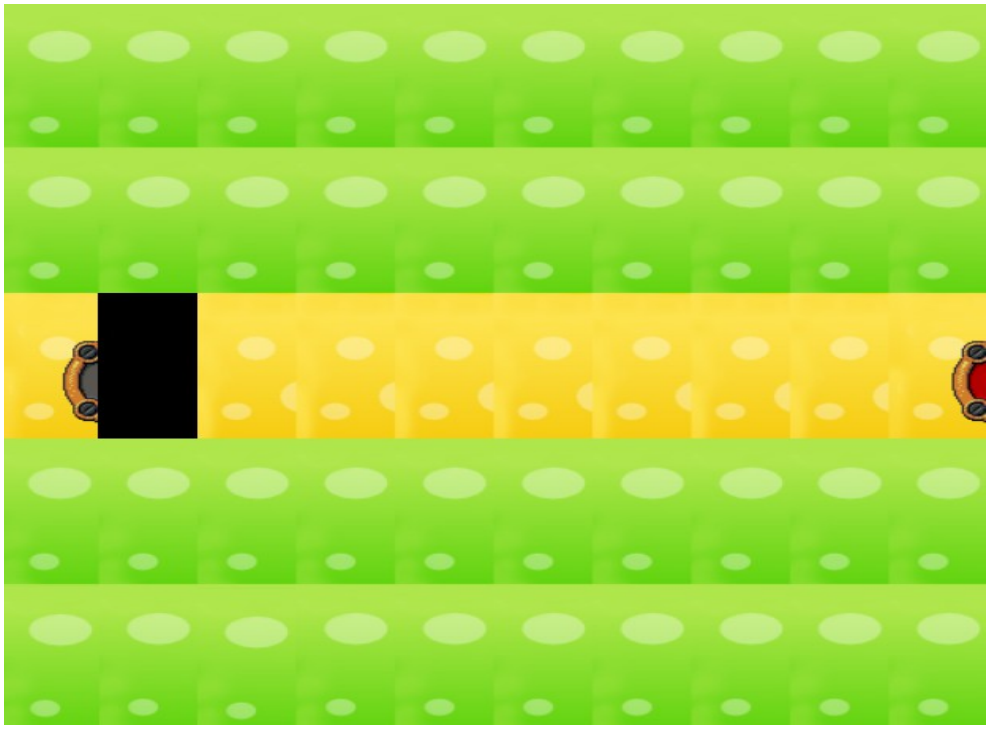
Travail réalisé :

Nous avons revu l'algorithme de déplacement des ennemis, ceux-ci se déplacent désormais. Le jeu est un peu plus joli. Le placement des tours est maintenant correctement pris en compte à l'aide du clic gauche de la souris. Les ennemis et les tours ont leur couleur respective.

Problèmes Rencontrés :

- Le joueur peut poser des tours à l'infini et les superposer
- Les ennemis ne se déplacent pas sur toute la longueur du chemin.

État du jeu :



Semaine 9 : Du 12/01 au 18/01

Objectifs:

- faire bouger les ennemis
- changer l'image de l'ennemi et de la tour
- implémenter le tir des tours sur les ennemis

Travail réalisé :

Encore un changement de l'algorithme de déplacement des ennemis mais sans succès, Création de la classe Bullet qui doit permettre d'avoir une balle et de la déplacer sur l'ennemi . Les ennemis ressemble à des chars, et les tours à des personnages verts.

Problèmes Rencontrés :

- Le joueur peut poser des tours à l’infini et les superposer
- Les ennemis ne se déplacent pas sur toute la longueur du chemin.
- Les tours ne tirent pas encore.

Semaine 10 : Du 19/01 au 25/01

Objectifs :

- Redéfinition de certaines fonctions
- ajouter des fonctions au jeu

Travail réalisé :

La classe ImageHandler, qui gère le chargement des textures du jeu, permet maintenant de rajouter plus facilement de nouveaux éléments dans le fichier texte des niveaux. On peut maintenant mettre le jeu en pause, et l'implémentation de la victoire et de la défaite sont commencés.

Problèmes Rencontrés :

- Le jeu indique qu'il est en pause mais continue de jouer.
- Le changement de niveau se fait aléatoirement.

Semaine 11 : Du 26/01 au 01/02

Objectif :

- Correction des bugs

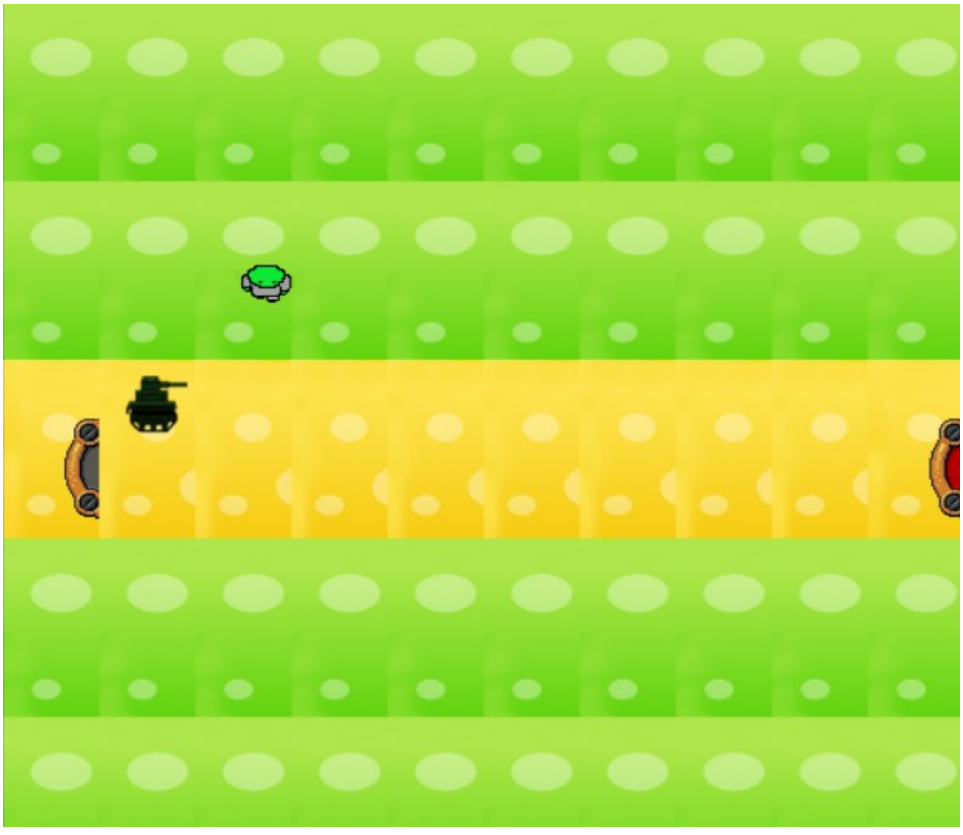
Travail réalisé :

La pause fonctionne correctement. Les conditions de changement de niveau sont précisées. Les cas de victoire et de défaite sont terminées mais non testées.

Problème Rencontré :

- Sans le déplacement complet de l'ennemi et sans le tir de la tour, les fonctions de victoire/défaite/changement de niveau ne sont pas testées dans les conditions réelles.

État du Jeu :



Semaine 12 : Du 02/01 au 08/02

Objectifs :

- Faire avancer l'ennemi
- Faire tirer les tours

Travail réalisé :

L'ennemi ne bouge toujours pas, nous envisageons d'abandonner notre première solution et de repartir de zéro. Les tours ne tirent pas non plus.

Problème Rencontré :

- l'algorithme de déplacement semble trop long et trop complexe pour un simple déplacement.

Semaine 13 : Du 09/01 au 15/02

Objectifs :

- Faire avancer l'ennemi
- Faire tirer les tours

Travail réalisé :

Changement d'algorithme pour le déplacement. Nous avons décidé de passer à quelque chose de plus simple à l'aide de point de passage. Les ennemis se déplacent et nous avons pu tester que la défaite est bien prise en compte.

Semaine 14 : Du 16/01 au 22/02

Objectif :

- Faire tirer les tours

Travail réalisé :

Les tours tirent bien sur les ennemis, les ennemis meurent correctement. La condition de victoire a été corrigée, le changement de niveau se passe correctement.

Semaine 15 : Du 23/01 au 01/03

Objectif :

- Ajout de fonctionnalités.

Travail réalisé :

Ajout de la gestion de l'argent pour éviter la pose infinie de tours. Ajout de la gestion de vie pour ne pas perdre dès qu'un ennemi atteint le point d'arrivée. Ajout d'un affichage de la vie et de l'argent.

État du Jeu :