Solène Mary-Vallée
Baptiste Ory

Projet IMAC Tower Defense - Flower Tower

/Projet C IMAC 1 S2

Programmation et algorithmique, Synthèse d'image
/Steeve Vincent, Olivier Falconnet et Venceslas Biri

/Sommaire

/Introduction

/Présentation du projet

Le projet qui nous a été confié pour les cours de « Programmation et algorithmique » et « Synthèse d'image » était la réalisation d'un **jeu de tower defense**.

Ce programme devait ainsi pouvoir **charger une carte, faire déplacer des vagues de monstres en fonctions de cette carte et permettre d'ériger des constructions pour détruire ces monstres**.

Il nous a été proposé d'utiliser **le format d'image PPM** (portable pixmap) pour le chargement de l'image de la carte et créer **un fichier .itd** (IMAC tower defense) pour la vérification et l'implémentation de cette même carte. Cela devait aussi être l'occasion d'expérimenter **les graphes et des algorithmes complexes** tel que l'algorithme de Dijkstra (pour les plus courts chemins).

On a choisis d'appeler notre projet « **Flower Tower** » et de l'ancrer dans une esthétique et une ambiance flower power des années 60-70.

/Mise en contexte

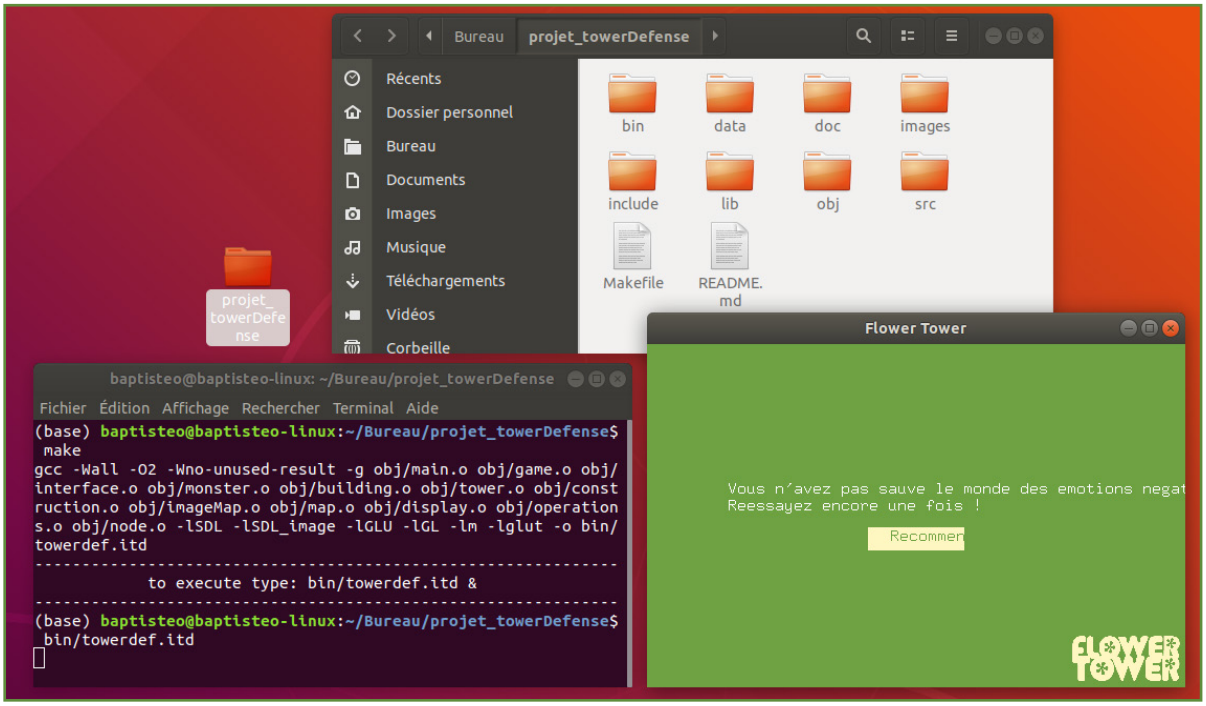
Le lien **github** du projet est : https://github.com/BaptisteOry/projet_towerDefense.

Voici la **structure des répertoires du projet** :

```
projet_towerDefense/
    \-- bin/ -> exécutable (towerdef.itd)
    \-- include/ -> fichiers .h
    \-- lib/ -> librairies (freeglut-3.0.0)
    \-- data/ -> fichiers carte .itd (map01.itd)
    \-- images/ -> images et sprites du jeu
    \-- doc/ -> documentation (sujet et rapport)
    Makefile -> compilateur
    README.md -> lisez-moi
```

Le **projet exécutable** (depuis la racine du dossier du projet) est : bin/towerdef.itd.

Par ailleurs, un fichier **Makefile** permet de compiler le projet (commande make). Lire le fichier **README.md** (à la racine du projet) pour plus d'informations.



Mise en contexte du fonctionnement du programme sur Linux (Ubuntu)

/Structure et algorithme

/Architecture de l'application

> main.c

Affichage (vue), actions utilisateur (contrôleur) et événements du jeu : **centralise** tout pour faire fonctionner le programme.

> interface.c

Gère l'**interface**. Prend en charge les différentes fenêtres, boutons et affichages textuels rencontrées au cours du jeu (début, fin, aide, argent, numéro de vague, etc.).

> display.c

Gère l'**affichage** en général (textes, textures et formes).

> game.c

Gère la **partie** : vagues de monstres, mort des monstres, fin de la partie.

> map.c

Contient les fonctions de **vérifications d'une carte** (grâce au fichier .itd et à l'image .ppm).

> imageMap.c

Gère ce qui est **relatif à l'image .ppm pour la carte**, notamment la lecture d'un fichier .ppm.

> node.c

Contient des fonctions relatives aux **noeuds** (graphe) pour implémenter une carte. Cela permet notamment à partir de la liste des liens récupérée sur le fichier .itd de former le graphe en ajoutant la liste des noeuds liés.

> monster.c

Gère les **monstres**. Il y a des fonctions relative à la liste chaînée des monstres (allouer, ajouter, détruire, etc.) et d'autres relatives à leurs déplacements.

> tower.c

Gère les **tours** avec des fonctions sur la liste chaînée associée (allouer, ajouter, détruire, etc.), d'affichage des informations et de sélection.

> building.c

Gère les **bâtiments** avec des fonctions sur la liste chaînée associée (allouer, ajouter, détruire, etc.), d'affichage des informations et de sélection.

> construction.c

Centralise des éléments propres aux **tours et bâtiments** : effets de bonus, placement par rapports aux autres construction (tours et bâtiments) et par rapport aux chemins.

> operations.c

Contient les **fonctions mathématiques**, notamment pour les intersections de forme, l'aléatoire et l'algorithme de Dijkstra.

/Fichier .itd

Un **fichier .itd** (dans /data) permet d'implémenter une carte dans le jeu (avec chargement et vérification et par map.c et imageMap.c). Voici la structure de notre fichiers .itd pour la première carte :

Mot clé	Valeur
carte	map01.ppm
chemin	0 255 0
noeud	255 0 0
construct	255 255 255
in	255 0 255
out	255 255 0

Il y a aussi les informations et liens de chaque noeud (graphe).

/Contenu

/Fonctionnalités

Fonctionnalités	Description
Lecture et vérification .itd	Lecture du fichier .itd. Vérification du format du fichier et stockage des informations nécessaires. (map.c et node.c)
Lecture et vérification carte .ppm	Lecture de la carte .ppm. Vérifier que la couleur de chaque pixel correspond bien à la zone qu'il détermine. (map.c et imageMap.c)
Graphe	Création du graphe des chemins selon les données du fichier .itd. (node.c)
Sprites et affichage	Création de l'interface et affichage des images et formes. (interface.c et display.c)
Gestion du jeu	Gestion des différentes phases de jeu : début, en cours, pause, fin. Quitter le jeu. Gestion des vagues de monstres. Mort des monstres. (game.c)
Gestion bâtiments et tours	Types et caractéristiques, création, destruction, placement, effets des bâtiments sur le tours et dégâts effectués par les tours sur les monstres. (tower.c, building.c et construction.c)
Gestion des monstres	Types et caractéristiques, naissance, déplacement, barre de vie. (monster.c)
Gestion du temps	Temps entre chaque vague. Pause : le temps n'est compté que lorsque la partie est en cours. (main.c et game.c)
Gestion de l'argent	Achat et vente de bâtiments et tours. Gain en tuant des monstres. (main.c et game.c)

Fonctionnalités supplémentaires	Description
Barre de vie des monstres	Barre de vie au-dessus des monstres relatives à leurs points de vie. Devient rouge si elle est trop basse. (monster.c)
Déplacement stochastique des monstres	Déplacement aléatoire des monstres à chaque noeud (en plus de Dijkstra) avec des restrictions. (monster.c)
Affichage de texte et de fenêtres	Affichage de textes grâce aux librairies glut et freeglut. Affichage de fenêtre (avec les textes) selon le statut du jeu : début, pause, fin, etc. (interface.c et display.c)
Visualisation du placement des constructions	Visualisation de la forme et de la constructivité des tours et bâtiments pendant leur placement. (main.c, tower.c et building.c)

/Règle du jeu

Dans le jeu, il faut survivre à toutes les vagues d'émotions négatives (les monstres) en plaçant des fleurs (les tours) et en les améliorant grâce à des combis (les bâtiments).

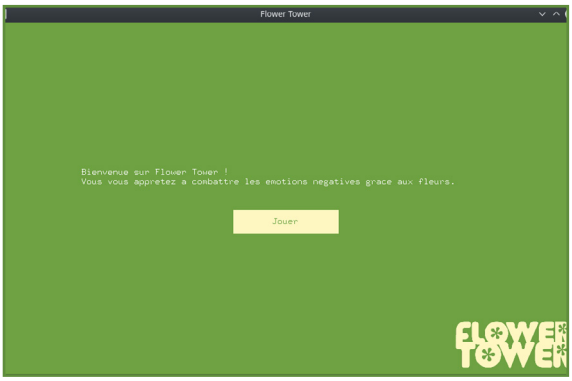
Voici les **commandes utilisateur** du jeu :

> H : aide et pause
> Echap : quitter
> A / Z / E / R + clic gauche : construire une fleur
    > A : fleur rouge - puissante
    > Z : fleur violette - équilibrée
    > E : fleur jaune - cadence rapide
    > R : fleur bleue - longue portée
> Q / S / D + clic gauche : construire un bâtiment
    > Q : combi radar - augmente la portée des tours
    > S : combi usine - augmente la puissance des tours
    > D : combi stock - augmente la cadence de tir des tours
> Clic gauche sur construction : consulter les informations
> Clic droit sur construction : détruire une construction

Des **fenêtres correspondantes au statuts du jeu** (début, pause, fin, etc.) avec des boutons aiguillent également l'utilisateur pour des actions de déroulement du programme

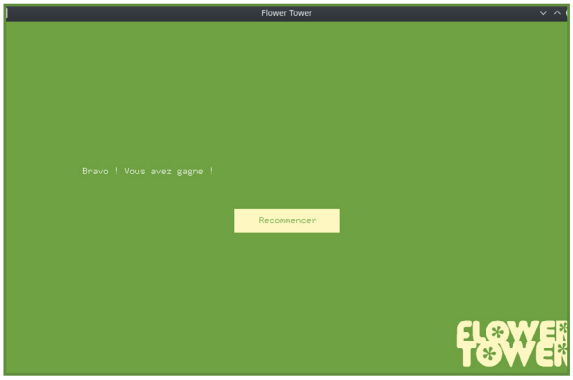
/Résultats

/Fenêtres de statuts

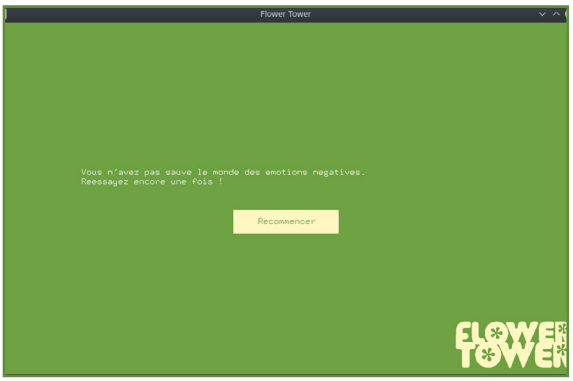


Accueil

Pause et aide

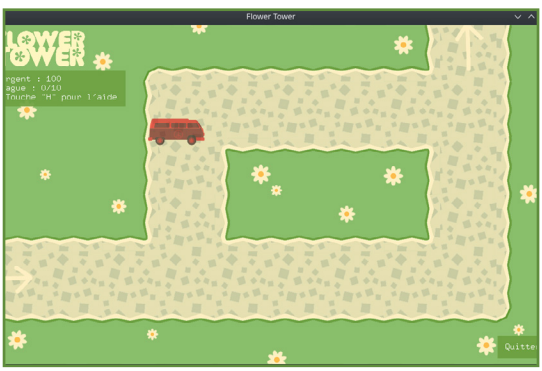


Victoire

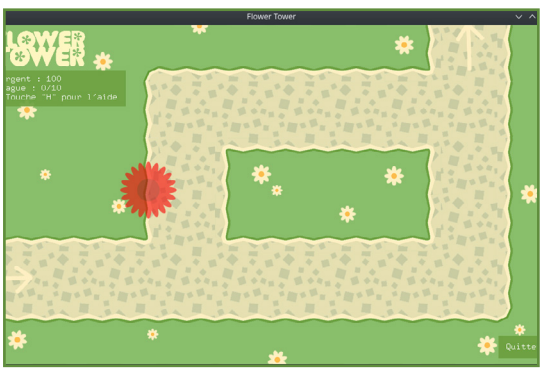


Défaite

/Jeu

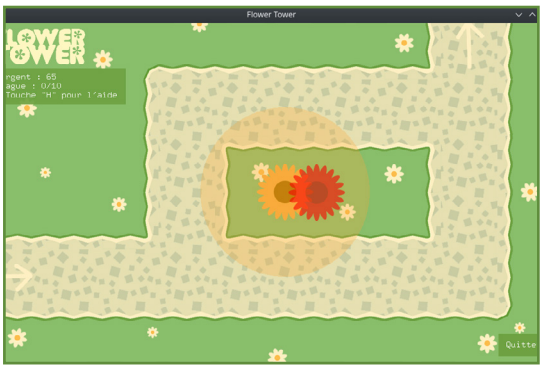


Placement des bâtiments 1

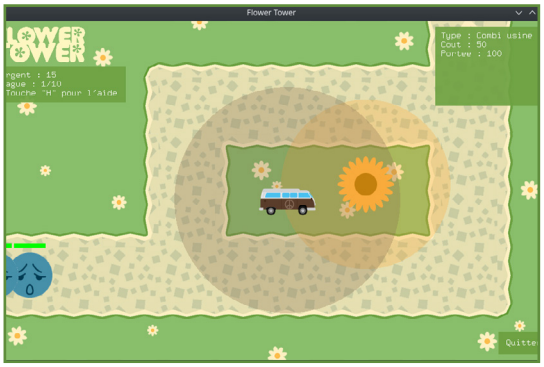


Placement des tours 1

Placement des bâtiments 2

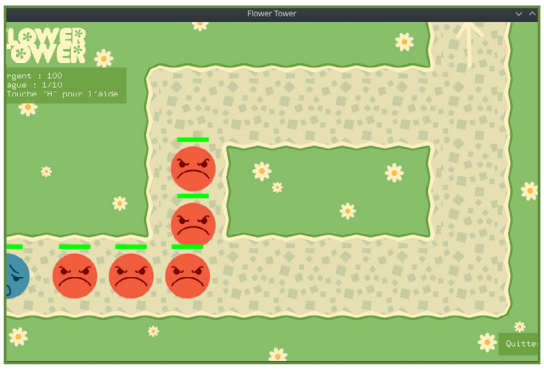


Placement des tours 2

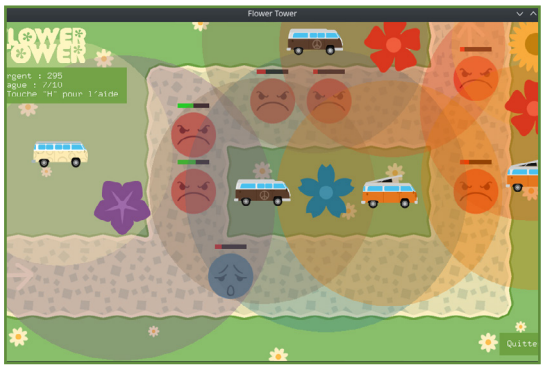


Infos d'un bâtiment

Infos d'une tour



Arrivée d'une vague de monstres



Partie avancée

/Conclusion

Ce projet fut un bon exercice pour **appliquer tout ce qu'on avait vu en cours** (graphes, Dijkstra, librairie SDL, structures de données, etc.).

Nous regrettons de ne pas avoir pu coder le programme en **langage C++**. En effet, cela nous aurait permis de simplifier notre programme en faisant des héritages par exemple (en priorité avec les tours et les bâtiments). Nous aurions aussi aimé **améliorer quelques-unes de nos fonctionnalités ou en implémenter d'autres** : niveaux (avec d'autres cartes), musiques, courbe de progression, monstres additionnels, projectiles ou encore affichage optimisé des textes.

Cependant, nous sommes fier de présenter ce projet qui nous semble **abouti pour une première version, propre et répondant convenablement au sujet**.

/Sources

Auteurs Giant Bomb. Tower Defense : Giant Bomb [en ligne]. Disponible sur : https://www.giantbomb.com/tower-defense/3015-413/.

Auteurs MDN. Détection de collisions en 2D : MDN web docs [en ligne]. 23/03/2019 (modif.). Disponible sur : https://developer.mozilla.org/fr/docs/Games/Techniques/2D_collision_detection.

HERRELAN, Clement. Le pathfinding avec Dijkstra : OpenClassrooms [en ligne]. 21/05/2019 (modif.) Disponible sur : https://openclassrooms.com/fr/courses/1125906-le-pathfinding-avec-dijkstra.

Illustrateurs Vecteezy : Vecteezy [en ligne]. Disponible sur : https://www.vecteezy.com/.

Kenney. Tower Defense (300 tiles/sprites) : OpenGameArt [en ligne]. 29/07/2016. Disponible sur : https://opengameart.org/content/tower-defense-300-tilessprites.

NOZICK, Vincent. Images, formats de fichier : Université Paris-Est Marne-la-Vallée [en ligne]. Disponible sur : http://www-igm.univ-mlv.fr/~vnozick/teaching/slides/M1_ti/06%20formats.pdf.

THOMPSON, Jeffrey. Collision detection : Jeff Thompson [en ligne]. Disponible sur : http://jeffreythompson.org/collision-detection/.

THYSSEN Anthony. ImageMagick v6 Examples - Common Image Formats : ImageMagick [en ligne]. 13/02/2012 (modif.). Disponible sur : http://www.imagemagick.org/Usage/formats/#pbmplus.

WETZEL, Charles. What is a «tower defense» game? : Charles Wetzel [en ligne]. Disponible sur : http://korea.charles-wetzel.com/Programs/tower_defense/tower_defense_games.htm.

Cours de « Synthèse d'image » et « Programmation et algorithmique » 2019, en IMAC 1 - semestre 2.

Lien github du projet : https://github.com/BaptisteOry/projet_towerDefense