Imac Tower Defense

Projet d'algorithmique et de synthèse d'image

21/05/2013

Katia Moreira & Audrey Guénée

Sommaire

Introduction

Le projet ImacTowerDefense a consisté en la création d'un prototype de jeu vidéo s'inspirant des jeux de stratégie de type « tower defense ». Le but de ces jeux est de protéger un terrain en empêchant des vagues d'ennemis de le traverser, à l'aide de tours de fortification.
L'application permet donc notamment le chargement d'une carte, le déplacement des monstres et la construction de tours.
Manipulant de nombreuses images, le programme utilise des fonctions OpenGL et a été développé en C.

Présentation de l'application

Défendez les couleurs du mariage pour tous !
Empêchez la « Manif' pour tous » d'avancer en dressant sur le chemin de la méchante Boutin et de la vilaine Barjot les symboles de l'égalité et de la tolérance.
Gérer correctement votre temps et votre budget pour faire triompher la gentille Taubira.
Terrassez vingt vagues d'ennemis pour gagner la partie !

Exécuter l'application

Se placer à la racine de l'application. Taper la commande **`make`**. Taper ensuite la commande **`bin/itd`** suivie du nom du fichier itd (fichier de description de la carte lié à une image ppm).
Ex : **`bin/itd map.itd`**
Si vous voulez modifier la carte, placer votre propre fichier itd dans le répertoire data. Placez l'image ppm associée dans le répertoire images.

Architecture de l'application

Structure du programme

bin/

itd - *Exécutable de l'application*

data/

map.itd - *Fichier décrivant la carte du jeu*

doc/

MoreiraGueneelTD.pdf - *Rapport de projet*

towerDefense.pdf - *Sujet du projet*

images/

Images des étapes du jeu

game-over.jpg

game-win.jpg

menu.jpg

map.ppm - *Image de la map*

interface/ - *Images liées à l'interface du jeu*

buttons.png

figures.png

pause.png

hybride.png

laser.png

mitraillette.png

rocket.png

towers/ - *Images des tours*

hybride.png

laser.png

mitraillette.png

rocket.png

monsters/ - *Images des monstres*

boutin.png

barjot.png

include/ - *Fichiers d'entête .h*

- Color.h
- Game.h
- Map.h
- Monster.h
- Node.h
- Tower.h
- tools.h

obj/- *Fichiers objets .o*

src/ - *Fichiers .c*

- Color.c
- Game.c
- Map.c
- Monster.c
- Node.c
- Tower.c
- main.c
- tools.c

Makefile - *Fichier pour compiler l'application*

Organisation du code

Nous avons créé un fichier pour chaque élément du jeu (en réalité, un fichier .c et son entête .h correspondante).

Les fichiers **Color.h** et **Color.c** définissent la structure Color et une fonction permettant de créer une nouvelle couleur. Ces fichiers permettent la gestion des zones couleurs de la carte (entrée, sortie, zone constructible...).

Les fichiers **Game.h** et **Game.c** définissent la structure Game et des fonctions d'affichage du jeu : affichage du budget, du nombre de vagues envoyées, affichage d'un nombre en général (*posFigure* détermine la position du nombre en fonction de sa catégorie (dizaine, centaine...), *displayFigure* affiche alors le chiffre à la bonne position).

Les fichiers **Map.h** et **Map.c** définissent la structure Map et différentes fonctions propres à la carte du jeu. La fonction *loadMap* charge un fichier itd et appelle *createMap* qui vérifie la validité de celui-ci et crée le cas échéant la liste des nœuds de la carte. La fonction *drawPath* permet l'affichage du chemin.

Les fichiers **Monster.h** et **Monster.c** définissent différentes structures propres aux monstres et les fonctions permettant leur gestion : création d'un monstre, création d'une liste de monstres, ajout et suppression d'un monstre, dessin d'un monstre, dessin des listes de monstres, calcul des monstres à la portée d'une tour.

Les fichiers **Node.h** et **Node.c** définissent la structure Node et une fonction de création d'un nœud. Ces nœuds correspondent à ceux du fichier itd et définissent le chemin de la carte.

Les fichiers **Tower.h** et **Tower.c** définissent la structure Tower ainsi que les différents types de tours et les fonctions de gestion des tours : création d'une tour, vérification de la validité de l'emplacement de la tour (pas sur une autre tour ou sur une zone non constructible), détermination de la type de tour à construire en fonction du clic sur l'interface de jeu, affichage des tours, affichage des informations d'une tour au survol de celle-ci.

Les fichiers **tools.h** et **tools.c** ont été créés afin de rassembler différentes fonctions utiles au programme et qui ne dépendent pas d'une structure en particulier. L'entête ne définit donc pas de structure (d'où l'absence de majuscule). Le fichier tools.c implémente la fonction *loadTexture* permettant de charger une texture et de retourner son identifiant ainsi qu'une fonction permettant d'afficher une image en arrière plan et deux autres fonctions affichant une image à droite ou à gauche de la fenêtre.

Cette organisation du code a pour but de le fragmenter en unités logiques. Chaque fichier correspond à un élément précis du jeu (monstre, tour par exemple). Le but est de limiter le nombre de lignes du fichier **main.c** afin d'éviter d'obtenir un code trop long et indigeste.

Structures de données

Color

```
typedef struct color {
      unsigned int r;
      unsigned int g;
      unsigned int b;
}Color;
```

Structure de données simple définissant une couleur avec ses composantes de rouge, vert et bleu.

Game

```
typedef struct game {
      int start;
      int pause;
      int help;
      int win;
      int over;
      int budget;
      int nbListsSend;
      int nbListsKilled;
}Game;
```

Structure permettant de stocker l'état du jeu (commencé, en pause, gagné, perdu ou aide affichée) ainsi que le budget actuel, le nombre de listes de monstres envoyées et le nombre de listes de monstres tuées par le joueur.

Map

```
typedef struct map {
      char* image;
      Color pathColor;
      Color nodeColor;
      Color buildingAreaColor;
      Color inColor;
      Color outColor;
      unsigned int nbNodes;
      Node* listNodes;
}Map;
```

Structure qui stocke le nom de l'image ppm correspondant à la carte, les couleurs des différentes zones de la carte, le nombre de nœuds et la liste des nœuds (soit le

premier nœud de la liste).

Monster

```
typedef enum {
      BOUTIN, BARJOT
}MonsterType;
```

Stockage des deux types de monstre.

```
typedef struct monster {
      MonsterType type;
      float initialLife;
      float life;
      unsigned int resistance;
      int move;
      int speedDelay;
      int posX;
      int posY;
      struct monster* next;
      struct node* nextNode;
}Monster;
```

Structure définissant un monstre : son type, son niveau de vie initial, son niveau de vie à un instant t, sa résistance, son état de déplacement, sa vitesse, sa position en X et en Y, le monstre suivant et le prochain nœud à atteindre.
L'attribut move permet de savoir si le monstre peut se déplacer. L'unité de temps étant le $10^{ème}$ de secondes, un monstre peut au mieux se déplacer d'un pixel chaque $10^{ème}$ de secondes. Entre les deux types de monstres l'un va deux fois plus vite que l'autre. L'un se déplace d'un pixel tous les $10^{ème}$ de secondes, l'autre d'un pixel $1/10^{ème}$ de secondes sur 2.
Pour cela on fixe un speedDelay. Pour bouger l'attribut move doit être égal au speedDelay, si ce n'est pas le cas on incrémente ce premier. Lorsque l'on bouge un monstre on remet l'attribut move à 0. Pour le monstre le plus rapide move et speedDelay valent donc toujours 0. Pour le monstre le plus lent move vaut 0 ou 1, speedDelay vaut 1.

```
typedef struct monsterList {
      struct monster* root;
      int nbMonsters;
      int nbMonstersSend;
```

```
}MonsterList;
```
Structure permettant de stocker une liste de monstres (le premier monstre de la liste), le nombre de monstres composant la liste à un instant t et le nombre total de monstres envoyés.

```
typedef struct monsterLists {
	MonsterList* lists[NB_MONSTER_LIST_MAX];
	int nbLists;
}MonsterLists;
```
Structure contenant un tableau de listes de monstre dont la taille vaut `NB_MONSTER_LIST_MAX` (une constante définit dans le fichier Monster.h), et le nombre de listes dans le tableau à un instant t.

```
typedef struct monsterToReach {
	int distance;
	Monster* monster;
	int listNum;
	struct monsterToReach* next;
}MonsterToReach;
```
Structure stockant les informations des monstres à portée d'une tour : sa distance à la tour, un pointeur vers le monstre, le numéro de la liste à laquelle le monstre appartient et un pointeur vers le monstre à portée suivant.
Cette structure permet de comparer les distance séparant les monstres de la tour afin de ne tirer que sur le monstre le plus proche.

Node

```
typedef struct node {
	int x;
	int y;
	struct node* next;
}Node;
```
Structure contenant la position en X et en Y d'un nœud et un pointeur vers le nœud suivant.

Tower

```
typedef enum {
	ROCKET, LASER, MITRAILLETTE, HYBRIDE, EMPTY
}TowerType;
```

On stocke les quatre types de tours. Le type **EMPTY** permet, au début du jeu, de signifier qu'aucune tour n'a encore été sélectionnée.

```
typedef struct tower {
      int posX;
      int posY;
      TowerType type;
      unsigned int puissance;
      unsigned int reach;
      unsigned int cadence;
      int price;
      struct tower* next;
}Tower;
```

Structure permettant de stocker les informations concernant une tour : sa position en X et en Y, son type, sa puissance, sa portée, sa cadence, son prix et un pointeur vers la tour suivante.

Fonctionnalités de l'application

Le jeu s'ouvre sur une image d'accueil. Pour démarrer la partie l'utilisateur doit appuyer sur la **touche S**.

Règles du jeu

Le but du jeu est de tuer tous les monstres qui arrivent par vague de 10. Si l'un des monstres parvient à la sortie, le jeu est perdu. L'interface affiche le nombre de vagues de monstres envoyées, l'utilisateur gagne la partie au bout de 20 vagues éliminées. Le budget alloué à l'utilisateur en début de jeu est également affiché sur l'interface. Il évolue au cours de la partie en fonction des monstres tués, des tours achetées et/ou vendues.

Les tours

L'utilisateur sélectionne une tour en cliquant sur l'un des 4 boutons **Choisir** de l'interface.

Il peut ensuite placer la tour sur une des zones constructibles affichées en gris. Tant que l'utilisateur ne sélectionne pas une autre tour, le type de tour à construire est toujours le même.
L'utilisateur peut ensuite avoir accès aux caractéristiques d'une tour construite sur la carte (portée, cadence, puissance et prix) au **survol** de cette dernière.

Enfin, une tour peut être supprimée/vendue au **clic droit** sur l'une d'entre elle.

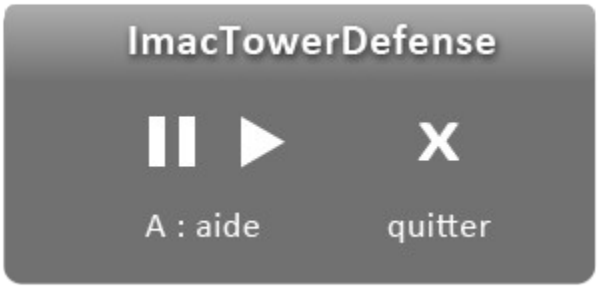
Les monstres

Chaque monstre tué rapporte une certaine somme d'argent qui permet à l'utilisateur de racheter des tours. A chaque nouvelle vague les monstres deviennent plus résistants. Deux types d'ennemis circulent sur la carte, ils diffèrent par leurs points de vie, leur résistance et leur vitesse.

Actions de l'utilisateur

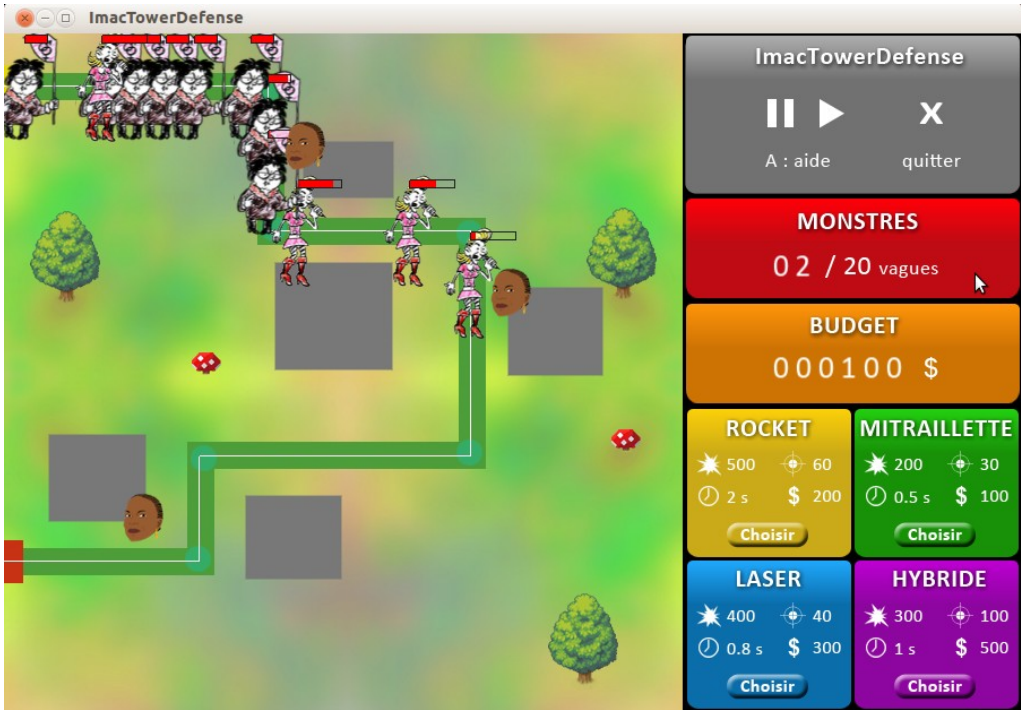
Outre le fait de pouvoir placer des tours sur la carte, l'utilisateur peut accéder à plusieurs fonctionnalités.

A tout moment il peut mettre le jeu en pause en cliquant sur le bouton pause et continuer le jeu en cliquant sur play. La fonction pause est aussi accessible grâce à la touche P. Lorsque le jeu est en pause, les monstres n'avancent plus et aucune tour ne peut être construite.
Il peut accéder à l'aide en appuyant sur la touche A.
Enfin, l'utilisateur peut quitter le jeu en cliquant sur la croix ou en appuyant sur la touche Q.

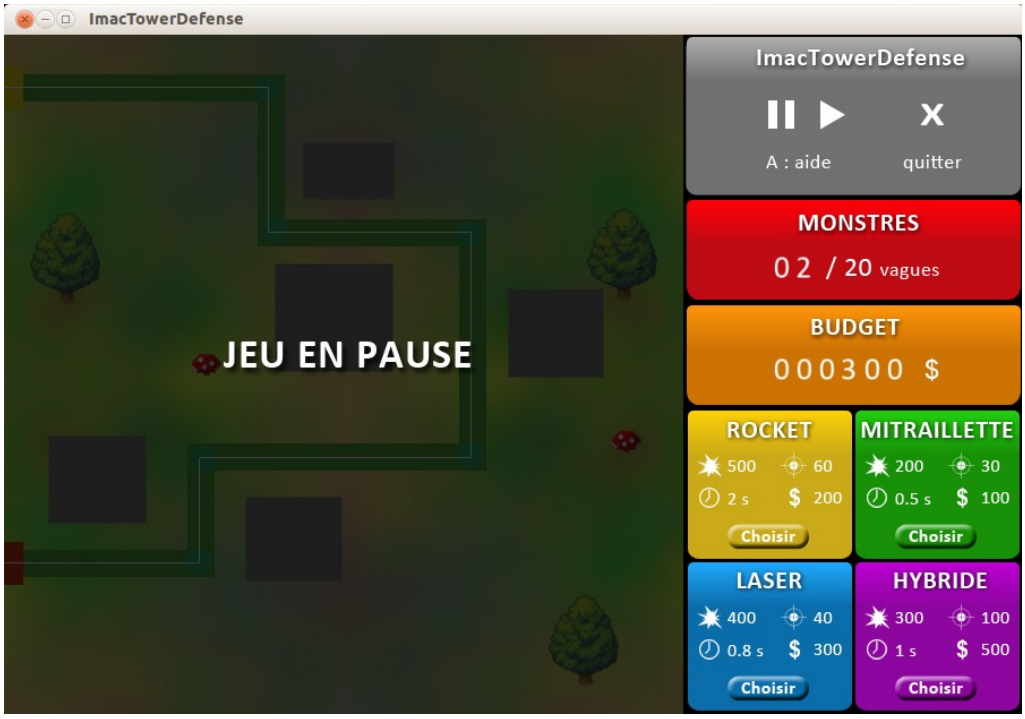


Résultats obtenus

Aperçu du jeu



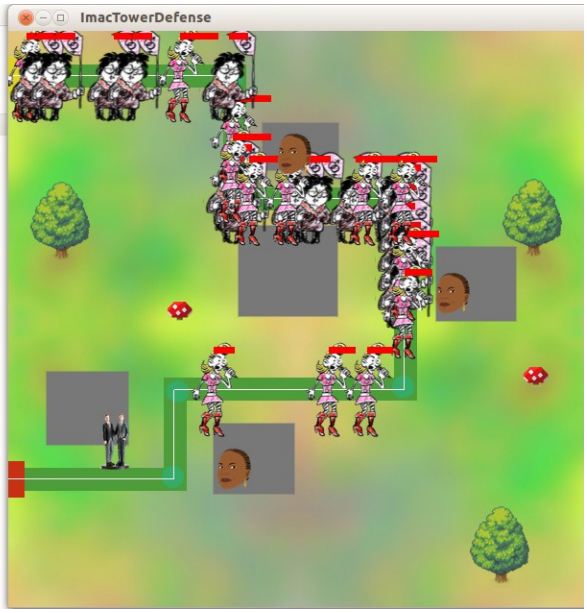
Aperçu du jeu en pause



Aperçu de l'aide

Le jeu consomme environ 22% du CPU lorsqu'il fonctionne à un régime important.

Process Name	User	% CPU	ID	Memory	Priority
indicator-messages-servic	audrey	0	1987	1.2 MiB	Normal
indicator-printers-service	audrey	0	1989	1.9 MiB	Normal
indicator-session-service	audrey	0	1991	1.1 MiB	Normal
indicator-sound-service	audrey	0	1988	1.2 MiB	Normal
itd	audrey	22	3686	142.6 MiB	Normal
mission-control-5	audrey	0	2132	1.0 MiB	Normal
nautilus	audrey	0	1896	14.0 MiB	Normal
nm-applet	audrey	0	1899	3.3 MiB	Normal
notify-osd	audrey	0	2232	2.9 MiB	Normal
plugin-container	audrey	0	2446	12.7 MiB	Normal
polkit-gnome-authenticati	audrey	0	1895	1.6 MiB	Normal
pulseaudio	audrey	0	1884	1.9 MiB	Very High
sh	audrey	0	1953	64.0 KiB	Normal
ssh-agent	audrey	0	1842	208.0 KiB	Normal
sublime_text	audrey	0	2540	17.2 MiB	Normal
sublime-text-2	audrey	0	2538	160.0 KiB	Normal
syndaemon	audrey	0	1876	184.0 KiB	Normal
telepathy-indicator	audrey	0	2101	1.8 MiB	Normal
ubuntu-geoip-provider	audrey	0	2030	1.0 MiB	Normal
ubuntuone-syncdaemon	audrey	0	2268	18.0 MiB	Normal
unity-applications-daemo	audrey	0	2044	2.6 MiB	Normal
unity-files-daemon	audrey	0	2046	1.1 MiB	Normal
unity-lens-video	audrey	0	2050	5.4 MiB	Normal
unity-music-daemon	audrey	0	2048	1.5 MiB	Normal
unity-musicstore-daemon	audrey	0	2135	628.0 KiB	Normal
unity-panel-service	audrey	0	1962	13.0 MiB	Normal



Conclusion

L'ensemble des fonctionnalités détaillées dans le cahier des charges ont été implémentées.

Des améliorations auraient toutefois pu être apportées, notamment concernant la difficulté du jeu (actuellement plutôt simple à gagner), les bonus suggérés et l'esthétique de l'interface.