Projet de programmation C

Shoot'em Up Gradius

KHELIFI Allan

Figure 1 – Shoot'em up réalisé inspiré de Gradius

Table des matières

Table des figures

Chapitre 1

Introduction

Le projet que nous avions à réaliser était le développement d'une application graphique à l'aide la librairie MLV, celle-ci en temps réel en manipulant le rafraichissement des images par secondes (FPS).

J'ai choisi de m'inspirer du segment Gradius© du jeu I Wanna Be The Boshy©, son principe de base étant le contrôle d'un vaisseau allié et quatre types d'ennemies différents apparaissant suivant différents patterns, le vaisseau allié possédant la capacité de tirer pour tuer ses ennemis qui, eux, essaient de tuer le vaisseau allié. Tout contact avec un ennemi par le vaisseau allié lui fait perdre des points de vie, de même qu'un contact avec un projectile ennemi tiré par un canon. Lorsqu'un ennemi meurt il peut générer un bonus qui, si ramassé par le joueur, lui permet d'obtenir différentes améliorations.

1.1 Les ennemis

1.1.1 Canon

Ce type d'ennemi est le seul type d'ennemi possédant la capacité de tirer. Son comportement est séparé en 3 étapes :

- Idle L'état dit "de base" où le canon ne tire pas.
- Angle 1 Le premier angle du canon, le canon tire à environ 45 degrés vers la gauche.
- Angle 2 Le deuxième angle du canon, le canon tire à environ 80 degrés toujours vers la gauche mais plus en haut.

Figure 1.1 – Sprites des ennemis Canon.

(a) Idle (b) Angle 1 (c) Angle 2 (d) Projectile

Les trois étapes prennent tout autant de temps l'une comme l'autre, avec donc autant de temps où le cannon ne tire pas (phase Idle), comme les phases où il tire - et celui-ci tirera 4 projectiles par phase de tir.

1.1.2 Lone Shot (Projectile seul)

Le projectile seul est un projectile au mouvement linéaire, lorsqu'un projectile seul apparait, il est suivi d'autres projectiles seuls. La particularité d'un projectile seul est qu'il possède une plus grande chance de générer un bonus lors de sa mort.

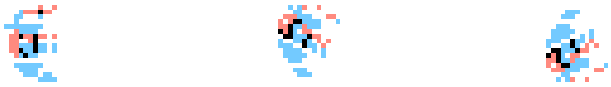
FIGURE 1.2 – Sprites des ennemis Lone Shot.

(a) Sprite 1 Lone shot (b) Sprite 2 Lone shot (c) Sprite 1 Bonus (d) Sprite 2 Bonus

1.1.3 Patterned

Les ennemis patterned sont des ennemis qui suivent un mouvement sous forme de pattern. Il possède 4 patterns qui permettent un mouvement en forme d'arche, les patterns sont décrits par un angle et une distance à parcourir sur cet angle.

FIGURE 1.3 – Sprites des ennemis Lone Shot.



(a) Sprite 1 Patterned (b) Sprite 2 Patterned (c) Sprite 3 Patterned

1.1.4 Spinning

Les ennemis patterned sont des ennemis qui suivent un mouvement en forme de sinusoidale. Tout en suivant leur trajectoire de sinusoidale, on les fait défiler sur plusieurs sprites donnant une impression qu'il tourne sur eux-mêmes.

FIGURE 1.4 – Sprites des ennemis Spinning.

(a) Sprite 1 Patterned (b) Sprite 2 Patterned (c) Sprite 3 Patterned (d) Sprite 3 Patterned

1.2 Le vaisseau joueur

Le joueur contrôle un vaisseau qu'il peut déplacer, il peut faire tirer et en fonction des bonus amassés améliorer avec les différentes améliorations possibles (voir Lone Shot (Projectile seul)).

FIGURE 1.5 – Sprites du vaisseau joueur.

(a) Idle (b) Moving up (c) Moving down (d) Projectile

1.2.1 Améliorations

Lorsque le joueur ramasse un bonus, un compteur croit et lorsque le joueur consomme son compteur, la n-ème amélioration est appliquée. Parmi ces améliorations :

FIGURE 1.6 – Sprites des améliorations.

(a) Projectile laser (b) Missile (c) Option

Chapitre 2

Manuel utilisateur

Le joueur peut déplacer le vaisseau à l'aide des flèches directionelles, tirer à l'aide de la touche espace (SPACE) (et évidemment tirer en se déplacant) et consommer les bonus amassés à l'aide de la touche control gauche (LCTRL).

Lorsque le joueur tue un ennemi, l'ennemi à une chance de générer un bonus à sa mort, qui sont les bonus permettant les améliorations du vaisseau joueur. Plus le joueur amasse de bonus, plus il dévérouille différentes améliorations (consommer les bonus sur une amélioration déjà acquise n'aura aucun effet) :

- Speed up Boost de vitesse.
- Missile Le vaisseau auto-tire des missiles vers le bas.
- Double Le vaisseau obtient un deuxième canon et tire deux fois dans la même direction.
- Laser Les tirs du vaisseau deviennent des projectile lasers qui font le double de dégâts. (se cumule avec l'amélioration Double)
- Option Le vaisseau obtient en option un objet qui le suit et reproduit les tirs du vaisseau.

La partie se continue tant que le joueur ne perd pas l'intégralité de sa vie, par défaut 100 points de vie et il en perd 10 à chaque contact avec un ennemi ou un projectile ennemi. Si touché, le joueur devient invulnérable pendant un peu plus d'une seconde.

Chapitre 3

Déroulement du programme

Le programme est englobé dans une boucle principale constituant la boucle de jeu. Cette boucle est faite de façon à pouvoir gérer le rafraichissement des images et maintenir le jeu à 60 images par seconde, pour le rendre le plus fluide possible.

Lors du déroulement de cette boucle est traité, à chaque passage, le déplacement de toutes les unités, ensuite sont résolues les collisions s'il y en à et enfin, actualise l'état de jeu en redessinant l'intégralité du jeu.

Le déplacement du vaisseau joueur dans un premier temps est effectué en vérifiant si des flèches directionelles sont enfoncées et en déplacant le vaisseau accordément.

Les unités ennemies elles, se déplacent de droite à gauche. Une fois arrivée sur le bord gauche, elles sont repositionnées à droite de l'écran et continuent sur leur trajectoire tant qu'elles sont vivantes.

Les projectiles, alliés comme ennemis, suivent leur vecteurs de déplacement.

Les collisions sont traitées en comparant les hitbox des différentes unités. On compare au vaisseau les hitbox des ennemis, des projectiles ennemis et enfin on compare les projectiles du vaisseau aux ennemis afin de résoudre les différentes collisions.

Une fois l'état de jeu actualisé, il est actualisé graphiquement. La partie se déroule tant que le joueur est en vie (s'est fait toucher moins de 10 fois).

Chapitre 4

Choix algorithmiques et difficultés rencontrées

4.1 Choix algorithmiques

4.1.1 Différentes listes utilisées

Lorsqu'une unité gère des projectiles (telle que le vaisseau allié ou les ennemis canons), ceux-ci utilisent la structure ShotList. Un objet ShotList contient une liste dynamiquement allouée d'un certain nombre de projectiles, un index qui est le dernier projectile inseré et une liste d'autant de booléens que la capacité de la liste.

L'algorithme d'insertion est relié au tri de cette même liste. Pour éviter beaucoup d'allocations de mémoire par nouveau projectile, on insère simplement les projectiles un par un en incrémentant l'index et en assignant respectivement les booléens. Si un projectile sort de l'écran ou entre en collision, il se voit supprimé de l'écran ce qui est traduit en mettant ses coordonnées à (-1, -1) et en désactivant le booléen dans la liste à sa position.

En accumulant les insertions, la liste se retrouve au bout d'un moment pleine. Dans le cas où tous ses projectiles sont actifs (rare, voire impossible), on ne peut plus ajouter car la liste est pleine. Autrement, si des projectiles sont inactifs (donc comme si la liste avait des éléments vides), on la trie à l'aide d'un qsort sur ses éléments, triés par leurs coordonnées. Les élements donc tous inactifs sont déplacés en fin de liste, ce qui laisse de la place vide. On déplace donc l'index à la position du premier élément inactif, pour pouvoir insérer par dessus tous ces éléments vides/inactifs.

FIGURE 4.1 – Différentes étapes/états de la liste lors d'insertion de projectiles.

(a) Etape 1 (b) Etape 2 (c) Etape 3

Ici, l'étape 1 pourrait correspondre à une situation où 3 ennemis ont été générés, le deuxième est mort mais le premier et le troisième sont encore en vie. L'index pointe donc sur 3, la position après le dernier ennemi. Insérer un ennemi ici impliquerait le déplacement de l'index sur la position 4 et ajouterait un ennemi dans cette 4e position (ce qui peut être illustré par exemple par l'étape 2, où un ennemi serait ajouté en 4 mais le premier ennemi serait mort).

Suivant depuis l'étape 2, ajouter un ennemi ici implique un arrangement de la liste puisque l'index est égal à la capacité. Une fois arrangée, la liste ressemblerait à la liste de l'étape 3.

Cette structure est aussi utilisé pour la liste d'ennemis et des bonus.

4.1.2 Appels systèmes

Les différentes images et sons utilisés sont chargés dans des variable globales au lancement du programme. Ceci est utilisé afin de limiter les appels systèmes (charger les images et sons) qui causaient beaucoup de ralentissement avec la version précédente qui chargait et libérait chaque image lors de l'affichage.

4.1.3 Déplacements Patterned

Les déplacements des ennemis pattern sont codés à la brute avec des objets Pattern qui sont un vecteur et une distance à parcourir sur cet angle. Ainsi, pour créer les 4 cycles (Patterns) des ennemis Pattern, on pose un angle et une distance pour chaque déplacement.

4.1.4 Animation des unités

En manipulant les images par secondes nous sommes capable d'animer les unités en faisant une suite d'images, de façon similaire à un GIF. Chaque sprite sera affiché pendant tant d'images et les différents sprites affichés en boucle, pour donner une impression d'animation constante (voir les ennemis et leurs sprites).

4.2 Difficultés rencontrées

4.2.1 Ralentissement (lag)

Comme vu dans la section Appels systèmes, dans ma première version, une fonction de dessin d'une unité chargait l'image au début de celle-ci, la dessinait et la libérait. Il m'a fallu beaucoup de recherches pour comprendre que les appels systèmes sont assez lourds pour la machine et que stocker chaque image et son dans des variables globales améliorerait grandement la vitesse du programme, avec l'idée d'une structure contenant toutes ces données une deuxième option.

4.2.2 Collisions

L'algorithme des collisions à été retravaillé plusieurs fois. Les différentes versions connaissant multiples bugs où certains projectiles, de façon inexpliquée, n'étaient plus en collision quand ils devaient l'être. La version actuelle détecte, en comparant les hitbox, si un objet est dans l'autre. Cette version permet plus de liberté sur les hitbox et évite que des projectiles qui visuellement ne devraient pas toucher leur cible, touchent celle-ci.

4.2.3 Git

Ce projet fut pour moi une introduction à Git et la découverte intégrale de celui-ci. J'ai rencontré quelques difficultés lors du branching et n'ai pas réussi à maitriser le potentiel plein de Git au delà de l'hébergement de fichiers (pas de travail en parallèle donc recourt à des simples push sans branching pour simplement y stocker les versions).

Chapitre 5

Bugs

5.1 Imperfection des collisions

Les hitbox étant codés comme un rectangle entourant les unités, une grande partie des pixels à l'intérieur de celles-ci ne correspond visuellement pas à l'image, j'ai opté pour un algorithme différent au niveau des collisions (voir collisions) permettant de limiter ces pixels "erreur" mais fait que certains projectiles passent à travers les ennemis.