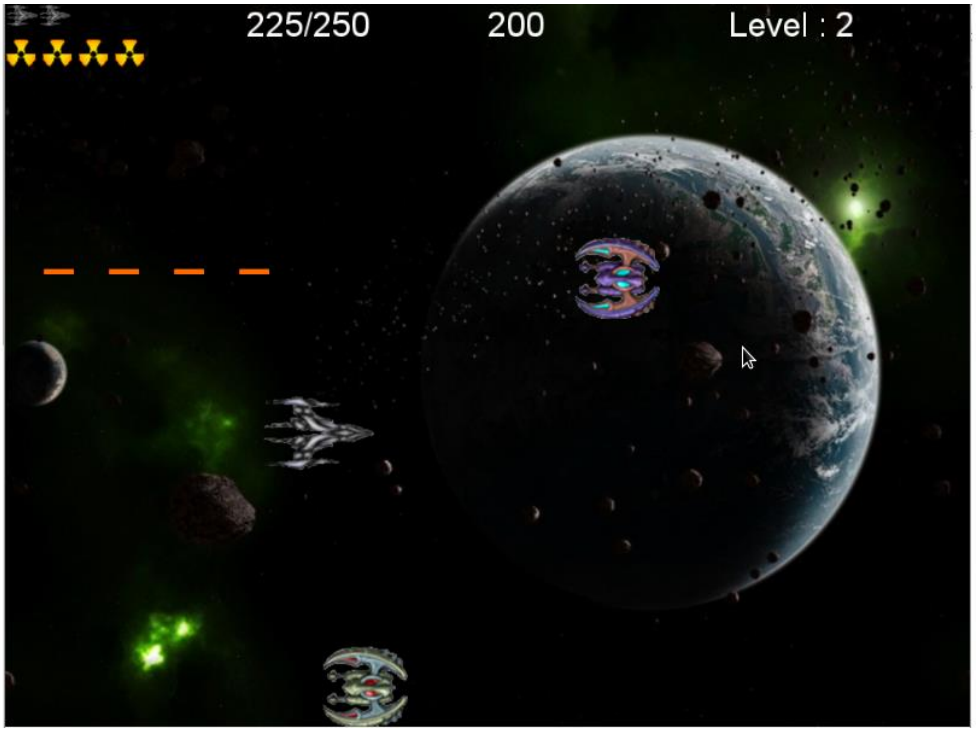
Rapport Shoot Them Up

Puyoou Aubin – Suner Arthur – Pavarino Arthur



IUT BORDEAUX 1

22 mai 2015

Rapport

Puyoou Aubin – Suner Arthur – Pavarino Arthur

Introduction

Dans le cadre du projet PT2, nous avions à réaliser en groupe un jeu de type « Shoot them up » dans le style du très connu « Space Invader ». Les objectifs principaux de ce projet étant principalement :

1. Mise en pratique des notions vues en M2103 Programmation Orientée Objet
2. Utilisation de la bibliothèque standard C++ (STL)
3. Intégration (i.e., utilisation) d'une bibliothèque externe -> la SFML
4. Travail en collaboration et autonomie de gestion de projet

Ce jeu devait être développer en C++ avec l'IDE Code::Blocks et être compilable et exécutable sur les machines de l'IUT. Nous avions 15 semaines pour le réaliser.

Le projet a été divisé en deux grandes parties

- La conception
- Le développement

Dans la partie conception nous avions réfléchis à l'architecture du jeu, aux problématiques et à organisation de celui-ci.

En transition, nous avons réalisé un premier livrable en console testant les fonctionnalités essentielles en s'appuyant sur la conception faite au préalable.

Dans la partie développement nous avons développez le jeu dans son intégralité et ajouter toute la partie graphique à l'aide de la bibliothèque SFML 1.6.

Mise en contexte

Un shoot them up pouvant se traduire par « dégommez les tous » est un style de jeu d'action et d'arcade où le joueur doit tirer sur un maximum d'ennemis sans mourir. Il dirige généralement un personnage ou un véhicule en se confrontant aux ennemis arrivant par vague.

Ce style de jeu est relativement ancien, en effet le premier du genre est apparu en 1962 avec « Spacewar ». C'est en 1978, avec l'arrivé de Space Invader que le genre décolle.

C'est dans les années 80 avec le développement des bornes arcades que le shoot them up est devenu populaire auprès du grand public.

L'équipe

Contrairement à la majorité des groupes, nous sommes une équipe de 3 personnes suite à une démission dans le groupe de TD et un accord avec Mr Bourqui. Faire le projet à 3 a été à la fois un atout mais aussi un défi au niveau organisationnel.

L'équipe complète se constitue donc de : Puyoou Aubin, Suner Arthur et Pavarino Arthur.

Rapport

Puyoou Aubin – Suner Arthur – Pavarino Arthur

Sommaire

Rapport

Puyoou Aubin – Suner Arthur – Pavarino Arthur

1) Architecture

Dans le cadre de ce projet nous devions organiser notre code sous la forme de l'architecture MVC. Dans la première partie du projet c'est-à-dire pendant la phase de conception nous avions organisé notre jeu simplement avec un modèle. Nous avions compris plus tard qu'il fallait intégrer une vue dans notre architecture.

Ci-dessous, nous avons mis le premier diagramme de classe que nous avons créé qui ne respecte donc pas l'architecture MVC. Nous avons par la suite modifié beaucoup d'éléments qui n'étaient pas corrects ou adaptés.

Nous avons fait la réadaptation nécessaire quand nous étions dans la partie développement du jeu et que nous avions des problèmes concrets d'architecture à traiter.

A l'heure actuelle, le jeu est architecturé avec une vue et un modèle.

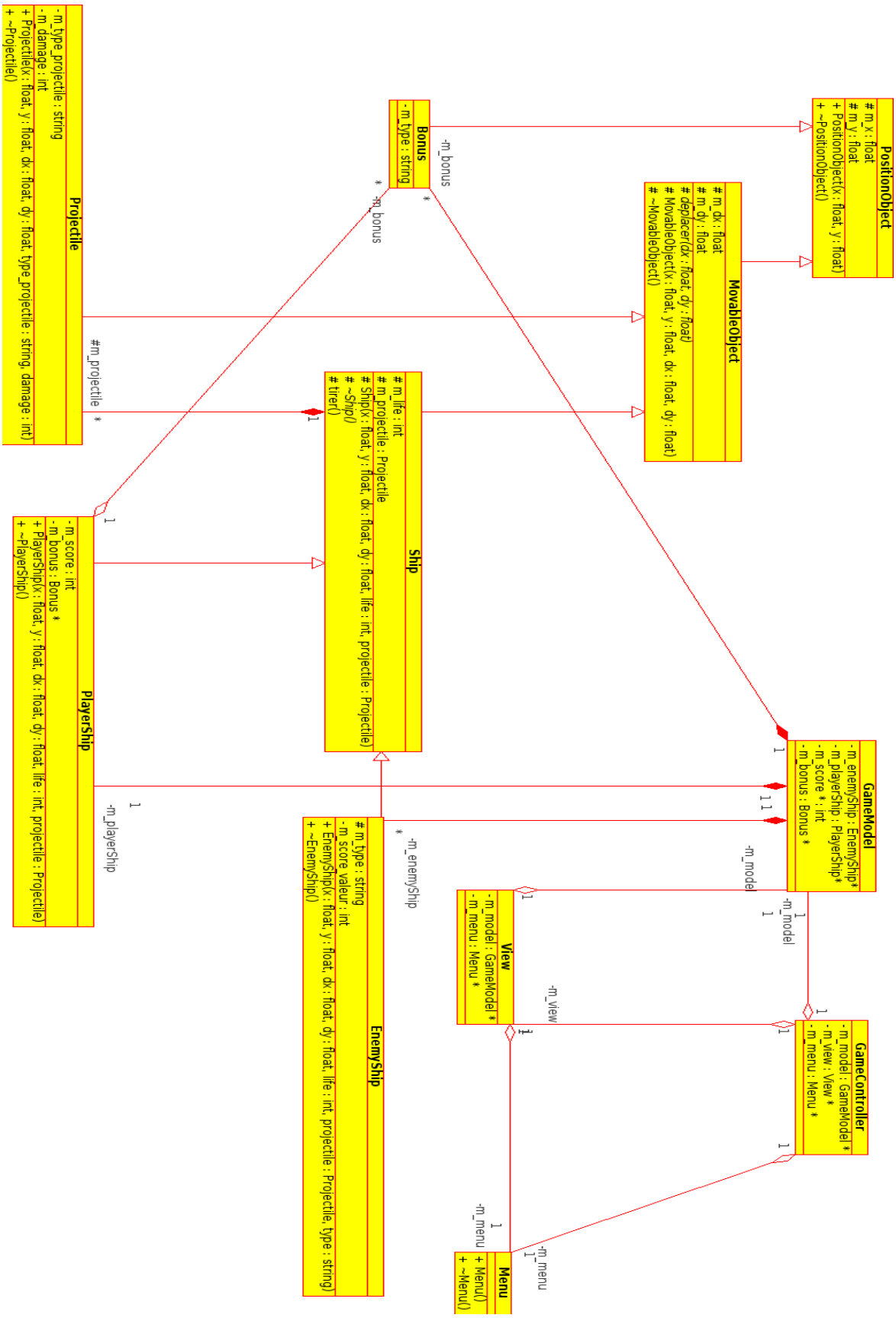
1) Modèle

Le modèle gère les interactions entre les éléments du jeu pour lequel l'utilisateur n'interagit pas. Il ne contient pas de SFML et gère une grosse partie du jeu. En effet, il gère le passage d'un niveau à un autre et il gère la difficulté du niveau puisqu'il contient un pointeur sur le niveau courant.

2) Vue

La vue gère tous les évènements externes (interactions de l'utilisateur) et l'affichage graphique de tout le jeu. C'est dans cette classe que se trouve la boucle principale du jeu. Cette classe possède aussi un pointeur sur le modèle courant du jeu qui permet de récupérer de nombreuses informations pour savoir quoi afficher à l'écran.

2) Diagramme de classe



Classe & héritage

Nous avons organisé nos classes en 3 niveaux distincts :

1) Niveau primaire

Nous avions tout d'abord mis en place deux classes, l'une abstraite et l'autre non, servant de socle à la majorité des autres classes.

- **PositionObject** : Cette classe sert à attribuer une position en X et Y. Nous avons créé cette classe en premier car nous savions dès le début du projet qu'il nous faudrait manipuler des coordonnées et des objets avec des coordonnées.

- **MovableObject** : Cette clase sert à attribuer un mouvement en X et Y. Elle hérite de PositionObject car nous savions aussi que la majorité de nos objets devraient se déplacer.

2) Niveau secondaire

Dans le niveau secondaire nous avons mis en place des classes abstraites correspondant à des éléments plus concrets dans le jeu mais étant toujours inutilisable.

- **Bonus** : Cette classe sert à définir des bonus dans le jeu puisqu'ils possèdent tous un effet unique. Comme les bonus se déplacent, elle hérite de MovableObject.

- **Ship** : Cette classe représente un vaisseau qui peut se déplacer ainsi que tirer. Elle hérite logiquement de MovableObject et est abstraite car le tir n'est pas défini de la même manière pour les ennemis que pour le joueur.

3) Niveau tertiaire

Dans le niveau tertiaire nous avons mis les classes instanciables qu'on utilise dans le GameModel.

- **PlayerShip** : Cette classe représente le joueur avec des méthodes de déplacements et de tirs adaptées pour ses interactions. Cette classe hérite de Ship.

- **EnemyShip** : Cette classe représente les ennemis, avec les méthodes de déplacements et de tirs aléatoires. Tout comme PlayerShip, elle hérite de Ship.

- **BonusBomb** : Cette classe représente le Bonus bombe qui permet de détruire tous les ennemis qui sont sur la map lorsqu'on appuie sur la touche ctrl. Le joueur peut

posséder un nombre infini de bombes. Ce bonus peut tomber d'un ennemi lorsqu'il est tué. Elle hérite de Bonus.

- **BonusHeal** : Cette classe représente le Bonus de vie qui permet de rendre le maximum de ses points de vie au joueur lorsqu'il est ramassé. Ce bonus peut tomber d'un ennemi lorsqu'il est tué. Cette classe hérite de Bonus.

- **BonusProjectile** : Cette classe représente le Bonus d'amélioration de tir du joueur qui permet d'augmenter son niveau de tir et ainsi faire plus de dégâts. Ce bonus peut tomber d'un ennemi lorsqu'il est tué. Cette amélioration peut s'appliquer jusqu'à un niveau de tir 3 et lorsque le joueur perd une vie, son niveau de tir retombe au niveau 1. Cette classe hérite de Bonus.

- **BonusShield** : Cette classe représente le Bonus de bouclier qui permet au joueur d'avoir un bouclier qui protège de 3 tirs ennemis. Ce bonus peut tomber d'un ennemi lorsqu'il est tué. Ce bonus se réinitialise lorsqu'un autre bonus de ce type est ramassé. Cette classe hérite de Bonus.

4) Autre classes

Nous avons également d'autres classes qui ne rentrent pas dans les catégories précédentes.

- **Levels** : Nous avons dans cette classe toute la gestion du niveau. Cette classe est instanciée dans GameModel et n'hérite d'aucune autre classe.

- **Projectile** : Cette classe permet de générer un projectile et de le gérer. Elle est instanciée depuis Ship. Elle hérite de MovableObject.

Présentation des algorithmes

1) Boucle principale

Dans le main, on créer un GameModel ainsi qu'un GameView. Ces deux structures possèdent les gros algorithmes à lancer à chaque tour de boucle (synchronize, nextStep, draw).

La boucle principale est directement en rapport avec la classe GameView. En effet, la fonction treatEvents gère le jeu et renvoie un booléen qui dit si le jeu est fini ou non. On fait tourner cette boucle tant qu'elle renvoie juste (tant que le joueur n'a pas quitté) et dans cette boucle, on fait les traitements nécessaires à chaque tour de boucle.

2) Algorithmes de jeu

Plusieurs fonctionnalités du jeu sont basé sur l'aléatoire notamment le déplacement des ennemis sur l'axe Y ou la fréquence de leurs tirs. On a également leur lieu d'apparition qui est géré aléatoirement ainsi que le type d'ennemis qui apparait.

Les bonus sont également affectés par l'aléatoire. En effet quand un ennemi est tué, le joueur à 33% de chance de récupérer un bonus.

On a un algorithme qui gère les collisions en récupérant l'origine des deux entités et en calculant si une entité rentre en collision avec une deuxième. Cela fonctionne de la manière suivante : On récupère l'origine en X et Y du premier élément. On test si le deuxième élément en X et supérieur ou égale à l'origine en X du premier. On teste également si le deuxième élément en X est inférieur ou égale à l'origine en X + la hauteur de l'entité et on fait pareille en Y mais en traitant à la fin avec la largeur au lieu de la hauteur.

C'est de cette manière qu'on détermine dans le menu si le joueur est sur un bouton ou non. Cette méthode de détection de collision est également utilisée dans le jeu pour détecter si un projectile du joueur touche un ennemi ou si un projectile ennemi touche les joueurs et ainsi faire les actions en conséquence.

Il y a aussi des petits algorithmes servant à garder le vaisseau dans l'écran afin qu'il ne dépasse pas. Des algorithmes supplémentaires permettent d'empêcher les ennemis de sortir de l'écran ou d'avoir des positions invalides, c'est-à-dire posséder des positions qui ne sont pas atteignables avec le tir du vaisseau du joueur.