Rapport Projet AP2 : Shoot Them Up

Théo CHASSAIGNE
Quentin HARSCOËT

2 juin 2013

Sommaire

Première partie
Rapport

1 Introduction

1.1 Présentation du projet

Un *Shoot Them Up* est un genre de jeu vidéo dans lequel le joueur contrôle un vaisseau ou un personnage qui peut tirer. Il doit détruire les ennemis qui arrivent vers lui. Le genre est né dans les salles d'arcade en 1978 avec *Space Invaders*.

Ce projet d'AP2 consiste à développer un Shoot Them Up, avec quelques spécificités imposées par les professeurs.

1.2 Objectifs

Le but de ce projet d'AP2 était de nous permettre de mettre en pratique les connaissance acquises dans la Programmation Orientée Objet pendant les cours à travers un exercice libre et ludique : la création d'un jeu vidéo.

Le jeu devait bien entendu utiliser un maximum des notions vues en cours (héritage, polymorphisme, transtypage) ainsi que la STL, à travers un PDF d'autoformation mis à notre disposition.

Ce projet avait aussi pour but de nous initier à une nouvelle bibiliothèque graphique, la SFML [1], qui est une bibliothèque orientée Objet, et donc prévue pour le C++, à la différence de la SDL, qui est à la base prévue pour être utilisée en C.

1.3 Déroulement

Ce projet s'est divisé en deux phases distinctes : une phase de conception, et une phase de réalisation.

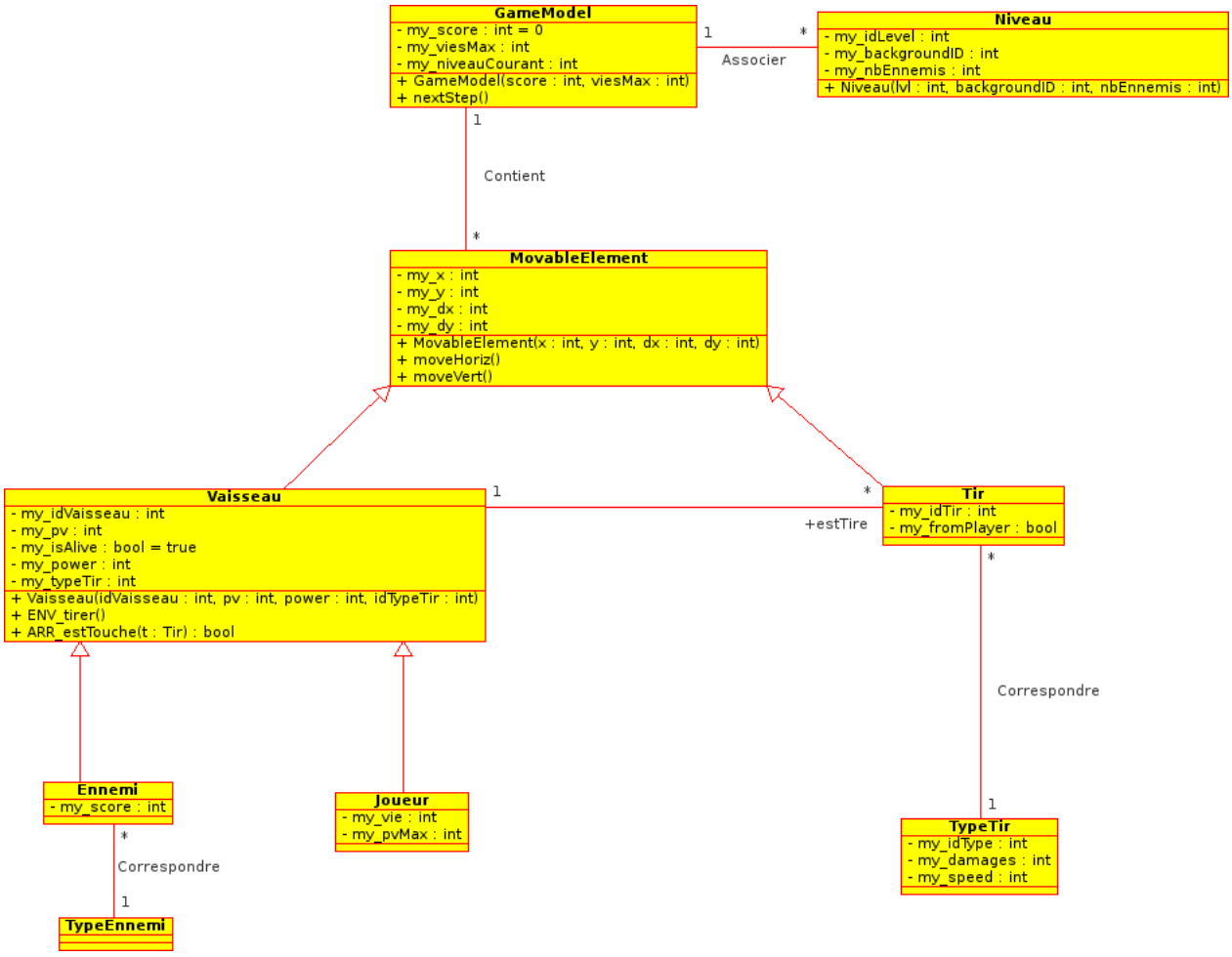
La phase de conception consistait à produire une spécification UML (Unified Modelling Language) du futur programme, en imaginant l'organisation des classes et les relations entre celles-ci.

La phase de réalisation corresspond à la programmation réelle du jeu, le but étant de respecter l'architecture définie pendant la phase de conception.

1. `http://www.sfml-dev.org/`

2 Présentation de l'architecture

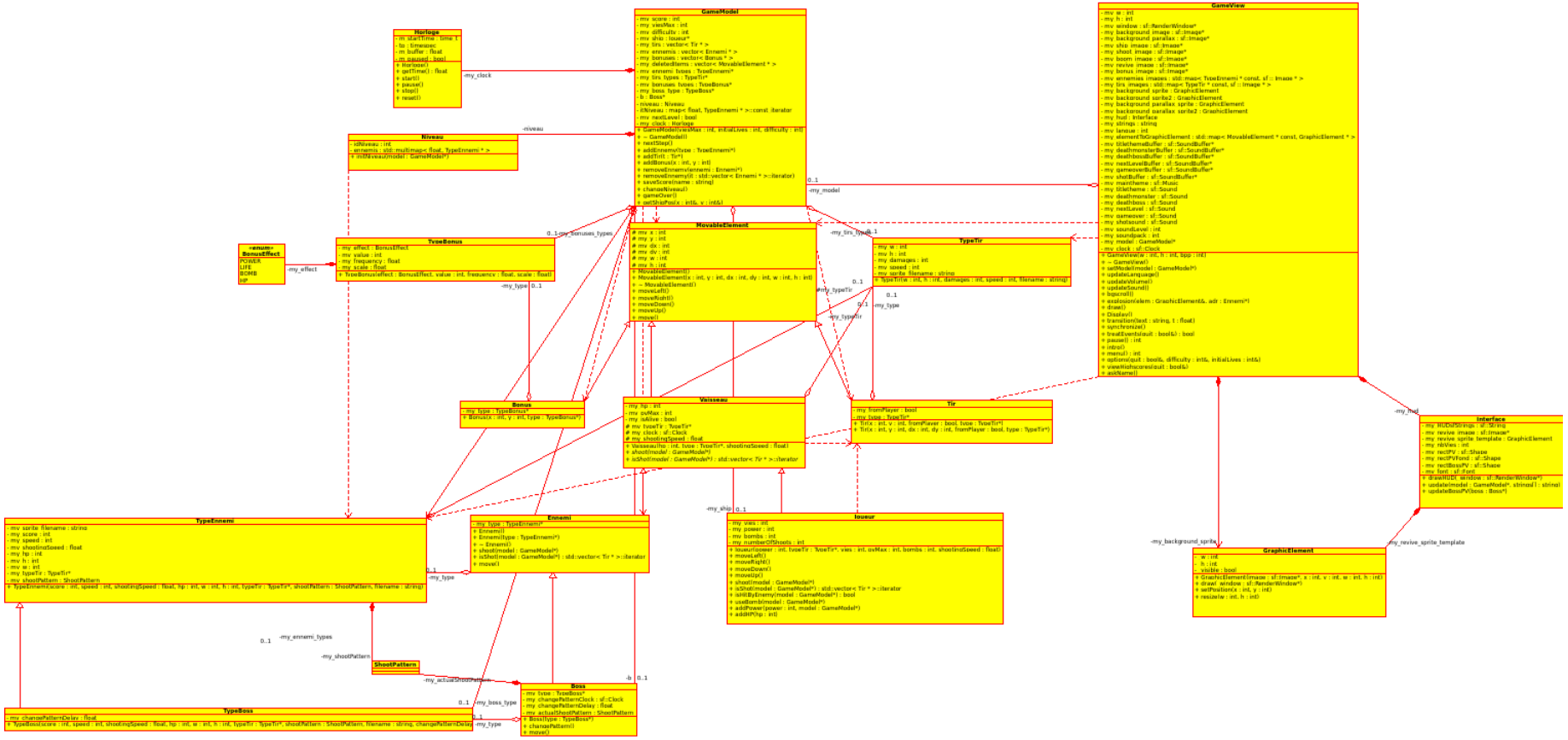
2.1 Architecture définie durant la phase de conception



2.2 Architecture finale

L'architecture finale a été générée à partir du code grâce à Umbrello [2]. Pour une visualisation facilitée, veuillez trouver le fichier Umbrello avec ce rapport.

2. `http://uml.sourceforge.net/`


Horloge
- m_startTime : time_t
- tic : timeseec
- m_buffer : float
- m_paused : bool
+ Horloge()
+ getTime() : float
+ start()
+ pause()
+ stop()
+ reset()
-my_clock
GameModel
- my_score : int
- my_vieuMax : int
- my_difficulty : int
- my_ship : loueur*
- my_tirs : vector< Tir * >
- my_ennemis : vector< Ennemi * >
- my_bonuses : vector< Bonus * >
- my_deletedItems : vector< MovableElement * >
- my_ennemi_types : TypeEnnemi*
- my_tirs_types : TypeTir*
- my_bonuses_types : TypeBonus*
- my_boss_type : TypeBoss*
- b : Boss*
- niveau : Niveau
- tNiveau : map< float, TypeEnnemi * >::const_iterator
- my_nextLevel : bool
- my_clock : Horloge
+ GameModel(vieuMax : int, initialLives : int, difficulty : int)
+ ~ GameModel()
+ nextStep()
+ addEnnemy(tvse : TypeEnnemi*)
+ addTir(t : Tir*)
+ addBonus(x : int, y : int)
+ removeEnnemy(ennemi : Ennemi*)
+ removeEnnemy(it : std::vector< Ennemi * >::iterator)
+ saveScore(name : string)
+ changeNiveau()
+ gameOver()
+ getShipPos(x : int&, y : int&)
-niveau
Niveau
- idNiveau : int
- ennemis : std::multimap< float, TypeEnnemi * >
+ initNiveau(model : GameModel*)
0..1
-my_model
GameView
- my_w : int
- my_h : int
- my_window : sf::RenderWindow*
- my_background_image : sf::Image*
- my_background_parallax : sf::Image*
- my_ship_image : sf::Image*
- my_shoot_image : sf::Image*
- my_boom_image : sf::Image*
- my_revive_image : sf::Image*
- my_bonus_image : sf::Image*
- my_ennemies_images : std::map< TypeEnnemi * const, sf::_Image * >
- my_tirs_images : std::map< TypeTir * const, sf::Image * >
- my_background_sprite : GraphicElement
- my_background_serie2 : GraphicElement
- my_background_parallax_sprite : GraphicElement
- my_background_parallax_serie2 : GraphicElement
- my_hud : Interface
- my_strings : string
- my_lanuse : int
- my_elementToGraphicElement : std::map< MovableElement * const, GraphicElement * >
- my_titlethemeBuffer : sf::SoundBuffer*
- my_deathmonsterBuffer : sf::SoundBuffer*
- my_deathbossBuffer : sf::SoundBuffer*
- my_nextLevelBuffer : sf::SoundBuffer*
- my_gameoverBuffer : sf::SoundBuffer*
- my_shotBuffer : sf::SoundBuffer*
- my_maintheme : sf::Music
- my_titletheme : sf::Sound
- my_deathmonster : sf::Sound
- my_deathboss : sf::Sound
- my_nextLevel : sf::Sound
- my_gameover : sf::Sound
- my_shotsound : sf::Sound
- my_soundLevel : int
- my_soundpack : int
- my_model : GameModel*
- my_clock : sf::Clock
+ GameView(w : int, h : int, fps : int)
+ ~ GameView()
+ setModel(model : GameModel*)
+ updateLanguage()
+ updateVolume()
+ updateSound()
+ bgscroll()
+ explosion(elem : GraphicElement&, adr : Ennemi*)
+ draw()
+ DisableVi
+ transition(text : string, t : float)
+ synchronize()
+ treatEvents(quit : bool&) : bool
+ pause() : int
+ intro()
+ menu() : int
+ options(quit : bool&, difficulty : int&, initialLives : int&)
+ viewHighscores(quit : bool&)
+ askName()
-my_hud
Interface
- my_HUDstrings : sf::String
- my_revive_image : sf::Image*
- my_revive_sprite_template : GraphicElement
- my_nbVies : int
- my_rectPV : sf::Shape
- my_rectPVfond : sf::Shape
- my_rectBossPV : sf::Shape
- my_font : sf::Font
+ drawHUD(window : sf::RenderWindow*)
+ updateImodel : GameModel* strinsuil : strinsl
+ updateBossPV(boss : Boss*)
-my_revive_sprite_template
-my_background_sprite
GraphicElement
- w : int
- h : int
- visible : bool
+ GraphicElement(image : sf::Image*, x : int, y : int, w : int, h : int)
+ draw(window : sf::RenderWindow*)
+ setPosition(x : int, y : int)
+ resize(w : int, h : int)
«enum»
BonusEffect
POWER
LIFE
BOMB
HP
-my_effect
TypeBonus
- my_effect : BonusEffect
- my_value : int
- my_frequency : float
- my_scale : float
+ TypeBonus(effect : BonusEffect, value : int, frequency : float, scale : float)
0..1
-my_type
0..1
-my_bonuses_types
MovableElement
my_x : int
my_y : int
my_dx : int
my_dy : int
my_w : int
my_h : int
+ MovableElement()
+ MovableElement(x : int, y : int, dx : int, dy : int, w : int, h : int)
+ ~ MovableElement()
+ moveLeft()
+ moveRight()
+ moveDown()
+ moveUp()
+ move()
0..1
-my_tirs_types
TypeTir
- my_w : int
- my_h : int
- my_damages : int
- my_speed : int
- my_sprite_filename : string
+ TypeTir(w : int, h : int, damages : int, speed : int, filename : string)
0..1
-my_typeTir
0..1
-my_type
0..1
-my_typeTir
Bonus
- my_type : TypeBonus*
+ Bonus(x : int, y : int, type : TypeBonus*)
Vaisseau
- my_hp : int
- my_pvMax : int
- my_isAlive : bool
my_tvseTir : TvseTir*
my_clock : sf::Clock
my_shootingSpeed : float
+ Vaisseau(hp : int, tvse : TvseTir*, shootingSpeed : float)
+ shoot(model : GameModel*)
+ isShot(model : GameModel*) : std::vector< Tir * >::iterator
Tir
- my_fromPlayer : bool
- my_type : TypeTir*
+ Tir(x : int, y : int, dx : int, dy : int, fromPlayer : bool, type : TypeTir*)
TypeEnnemi
- my_sprite_filename : string
- my_score : int
- my_speed : int
- my_shootingSpeed : float
- my_hp : int
- my_h : int
- my_w : int
- my_typeTir : TypeTir*
- my_shootPattern : ShootPattern
+ TypeEnnemi(score : int, speed : int, shootingSpeed : float, hp : int, w : int, h : int, typeTir : TypeTir*, shootPattern : ShootPattern, filename : string)
0..1
-my_type
Ennemi
- my_type : TypeEnnemi*
+ Ennemi()
+ Ennemi(type : TypeEnnemi*)
+ ~ Ennemi()
+ shoot(model : GameModel*)
+ isShot(model : GameModel*) : std::vector< Tir * >::iterator
+ move()
-my_ship
0..1
loueur
- my_vies : int
- my_power : int
- my_bombs : int
- my_numberOfShoots : int
+ loueur(score : int, tvseTir : TvseTir*, vies : int, pvMax : int, bombs : int, shootingSpeed : float)
+ moveLeft()
+ moveRight()
+ moveDown()
+ moveUp()
+ shoot(model : GameModel*)
+ isShot(model : GameModel*) : std::vector< Tir * >::iterator
+ isHitByEnemy(model : GameModel*) : bool
+ useBomb(model : GameModel*)
+ addPower(power : int, model : GameModel*)
+ addHP(hp : int)
0..1
-my_ennemi_types
-my_shootPattern
ShootPattern
-my_actualShootPattern
0..1
-my_boss_type
0..1
-my_type
Boss
- my_tvse : TvseBoss*
- my_changePatternClock : sf::Clock
- my_changePatternDelay : float
- my_actualShootPattern : ShootPattern
+ Boss(type : TypeBoss*)
+ changePattern()
+ move()
-b
0..1
TypeBoss
- my_changePatternDelay : float
+ TypeBoss(score : int, speed : int, shootingSpeed : float, hp : int, w : int, h : int, typeTir : TypeTir*, shootPattern : ShootPattern, filename : string, changePatternDelay : float)

2.3 Evolution de l'architecture

Nous pouvons remarquer que toutes les classes présente dans l'architecture de départ sont conservées dans l'architecture finale, même si les liens entre elles ne sont pas tout à fait les mêmes. On peut donc dire que nous avons plutôt bien respecté l'architecture que nous avions imaginé lors de la phase de conception.

3 Présentation des classes

Bonus : Classe représentant les différents bonus que l'on peut obtenir en tuant des ennemis. Est associé à un TypeBonus.

Boss : Les Boss du jeu, possède un délai de changement de pattern de tir qu'un ennemi classique ne possède pas.

Ennemi : Représente les ennemis, est associée à un TypeEnnemi.

GameMode : Modèle du jeu, contient et modifie tout les élements du jeu.

GameView : Vue, contient et modifie tout les élements graphques qui sont affichés à l'écran, et gère les évenements.

GraphicElement : Élélement graphique, hérite de la classe sf : :Sprite de la SFML.

Horloge : Mesure le temps, et propose une fonction de pause que la classe sf : :Clock de la SFML ne propose pas.

Interface : Contient tout les élements graphiques affichés pendant une partie (affichage du score, du niveau de vie, etc).

Joueur : Joueur, hérite de Vaisseau, et contient les éléments propres au joueur, comme le nombre de bombes.

MovableElement : Élément pouvant être déplacé, contient des attributs de bases comme une abscisse, une ordonnée, une longuer et une largeur.

Niveau : Un Niveau du jeu, contient un tableau associatif qui permet de faire apparaître les ennemis au cours du temps.

Tir : Un tir. Est associé à un TypeTir.

TypeBonus : Un type de bonus. Définit l'effet du bonus ainsi que sa valeur.

TypeBoss : Un type de boss. Une seule instance qui est actualisée à chaque changement de niveau.

TypeEnnemi : Un Type d'ennemi, spécifie ses caractéritiques de base, son TypeTir, ainsi que le chemin de son image.

TypeTir : Un Type de tir, contient la vitesse de déplacement du tir ainsi que les dommages qu'il inflige.

Vaisseau : Représente les différents vaisseau du jeu. Joueur et Ennemi en héritent.

4 Présentation de quelques algorithmes

4.1 Algorithme permettant aux ennemis de tirer vers le joueur

Ennemi.cc, lignes 61 à 72

```
1  int DY = model->getPlayerShip()->getY() - my_y; //Distance entre l'ennemi et le joueur
2  int DX = model->getPlayerShip()->getX() - my_x;
3
4  //Calcul du facteur entre la vitesse du tir et la distance entre l'ennemi et le joueur
5  float l = my_typeTir->getSpeed()/sqrt(pow(DX, 2) + pow(DY, 2));
6
7
8  float dy = l * DY;
9  float dx = l * DX;
10
11
12 Tir* t = new Tir(my_x, my_y, dx, abs(dy), 0, my_typeTir);
13
14 t->setX(my_x + my_w/2 - t->getW()/2);
15 model->addTir(t);
```

Le but de cet algorithme est de calculer le dx et le dy à appliquer au tir pour que celui-ci se déplace dans la direction du joueur à une vitesse égale à celle prévue par le tir.

Les deux premières lignes calculent la distance entre la position de l'ennemi qui tire et la position du joueur. Nous voulons que le tir aillent dans la même direction que si on lui donnait DX et DY, mais DX et DY le ferait se déplacer instantanément à la position du joueur.

Il nous faut donc calculer le facteur entre la distance qui sépare le tir du joueur, et la valeur de sa vitesse (qui correspond à la distance que l'on veut lui faire parcourir). C'est à ça que sert l, que l'on calcule en divisant la vitesse du tir par $\sqrt{DX^2 + DY^2}$

Enfin, on multiplie DX et DY par l, ainsi, on obtient dx et dy tels que $\sqrt{dx^2 + dx^2} = vitesse\ du\ tir$

4.2 Algorithme de génération des niveaux

Niveau.cc, lignes 11 à 26.

```
1  ennemis.clear();
2  float t;
3  for(int i = 0; i < 20 * (idNiveau+1) * model->getDifficulty() ; i++) // nombre d'ennemis     placer dans la
4  {
5          t = rand() % (15*idNiveau + 30);
6
7          int e = rand() % 3 ; //int qui d termine le type de l'enemi
8
9          ennemis.insert(pair<float, TypeEnnemi*>(t, model->getEnnemyTypes()[e]));
10 }
11
12 ennemis.insert(pair<float, TypeEnnemi*>(15*idNiveau + 30, model->getBossType()));
13 idNiveau++;
```

Cet algorithme est celui qui génére les niveaux. Un niveau est une représenté par une multimap associant un float à un TypeEnnemi : Le type ennemi spécifié apparait en jeu quand le temps associé est écoulé. L'utilisation d'une multimap est justifié par le fait que plusieurs ennemis peuvent apparaître en même temps.

Quand on entre dans la fonction, si l'on veut générer le niveau n, *idNiveau* vaut $n - 1$.

Pour générer la map au début de chaque niveau, on commence par supprimer son contenu. Puis, on rentre dans une boucle qui itère autant de fois qu'il y a d'ennemis à rajouter. Le niveau n contiendra $20 \times n \times difficulté$ ennemis.

On tire d'abord un nombre t compris entre 1 et $15 \times idNiveau + 30$, qui est la durée du niveau ainsi le niveau 1 dure 30 secondes, le niveau 2 en dure 45, etc.

Puis on tire un TypeEnnemi au hasard, puis on ajoute le couple temps/TypeEnnemi dans la multimap.

Enfin, on ajoute le boss à la durée maximale du niveau. TypeBoss héritant de TypeEnnemi, cette opération est possible.

5 Utilisation des notions vues en cours

5.1 Héritage

L'héritage joue un rôle prédominant dans le programme. En effet, une grande partie des éléments du modèle héritent de la classe MovableElement.

5.2 Polymorphisme

Le polymorphisme est utilisé lors du passage dans la vue, quand tout les éléments appartenant à des classes héritées de MovableElement sont tous dans une map¡MovableElement*, GraphicElement*¿ en tant que MovableElement.

Dans la classe Vaisseau, les méthodes *shoot* et *isShot* sont virtuelles pures, car un Joueur et un Ennemi ne tirent pas de la même façon, et ne sont pas affectés par les mêmes tirs (La méthode *isShot* dans Joueur ne renvoie que les tirs tirés par les ennemis, et celle des ennemis ne renvoie que ceux du joueur).

Les méthodes de MovableElement servant à se déplacer sont surchargées dans certaines classes filles, telles que Ennemi, ou Boss (qui se déplace différemment).

5.3 Transtypage

Le transtypage est utilisé pour faire la différence entre un Ennemi classique et un Boss dans certaines fonctions. Par exemple, dans GameModel : : nextStep(), quand on traite les ennemis, si l'ennemi est un boss, on peut lui appliquer la méthode *changePattern*, un boss ayant la particularité de changer de pattern de tir à intervalles réguliers.

6 Difficultés rencontrées

La principale difficulté était peu après le démarrage du projet. Une fois les classes de bases définies pendant la phase de conception implémantées, et l'obtention d'un vaisseau pouvant se déplacer, tirer et tuer les ennemis, comment gérer l'apparition des ennemis et la gestion des niveaux ?

Il était évident que l'apparition devait se faire en fonction du temps, le problème était la façon de l'utiliser. L'idée de la map associant un temps à un ennemi n'est venue que lorsque nous nous sommes dit que le but était de définir, pour un ennemi, à quel moment il devait appraître.

L'utilisation de TypeEnnemi* à la place d'Ennemi sert à économiser de la mémoire : stocker l'adresse d'un type existant est plus économe que de stocker un ennemi entier avec toutes ses caractéritiques.

Le fait que les ennemis ont une abscisse de départ aléatoire découle directement de ce fait.

L'autre problème induit par ce fonctionnement des niveaux est dans l'implémantations de la pause : les ennemis apparaissant en fonction du temps, il fallait être en capacité de mettre l'horloge du modèle en pause, sinon, l'horloge n'étant pas stoppée, tout les ennemis censé apparaître entre le moment où on aurait mis pause et le moment où l'on enlève la pause apparaîtrait tous d'un coup.

C'est ce fait qui a motivé la création de la classe Horloge, qui fournit donc une horloge capable d'être mise en pause. L'horloge ayant été réalisée uniquement avec les fonctions présentes dans le langage C++, elle n'a qu'une précision en secondes, du fait de l'utilisation de la fonction clock_gettime, qui remplit une structure de type timespec qui possède un champ tv_sec (l'utilisation du champ tv_nsec divisé par 1 000 000 renvoyant des valeurs bizarres).

7 Défauts du code

- L'un des défaut du code est le fait que tout les menus sont codés "en dur" dans la classe GameView, ce qui rend le fichier GameView.cc très volumineux. Il aurait peut-être été plus judicieux de créer une classe contenant tout ces menus, mais le menu de paramètre modifie des valeurs appartenant à GameView comme la langue ou le volume sonore, il aurait fallu lui passer les variables en paramètre, les menus étant appelés dans le main, cette solution n'aurait pas été très propres non plus.
- La fonction *shoot* de Ennemi, qui éxécute des bouts de codes différents en fonction du ShootPattern de l'ennemi en question. Il aurait été plus propre d'avoir une fonction shoot séparée pour chacun de ces pattern, mais dans ce cas il aurait fallu créer des classes ShootPattern au lieu d'une simple énumération, mais la manipulation des types d'ennemis et des ShootPattern qui leur sont associés aurait été plus difficile.

8 Qualités du code

La principale qualité du code est la gestion des types d'ennemis et des types de tir, qui sont très modulables. En effet, les types d'ennemis et les types de tirs sont définis dans le contructeur de GameModel, et rajouter un type d'ennemis ou de tir peut se faire en rajoutant simplement une ligne dans ce constructeur avec toutes les caractéritiques et le chemin de l'image associée, et en modifiant la constante correspondant au nombre de types d'ennemis ou de tir.

De même pour les ShootPattern : rajouter un ShootPattern revient à modifier l'enumération, et à rajouter la clause dans le switch de Ennemi : : shoot() (même si cela alourdit encore plus la fonction).

9 Conclusion

Ce projet aura donc été une réussite, dans le sens où il aura réussi sa mission de nous faire pratiquer la programmation par nous-même, en dehors des TP. Le fait que les consignes étaient moins précises que pour le projet de S1 a fait qu'un grand nombre d'aspect du jeu et l'organisation du codes est plus personnelle. On se rend compte en discutant avec les autres groupes que chacun a pris des orientations très différente, tant dans la partie programmation que dans les choix de gameplay.

Deuxième partie

Annexes

A Manuel d'utilisation du jeu

A.1 Démarrage

Au démarrage du jeu, on arrive sur l'écran titre :

- Jouer permet de lancer une partie
- Meilleurs Scores permet d'aller consulter les meilleurs scores
- Parametres permet d'accéder au paramètres et de changer la difficulté, la langue, etc...
- Quitter permet de quitter le jeu.

A.2 Déroulement d'une partie

Pendant une partie, voici à quoi ressemble l'écran

Sur cet écran sont affichés :
- Le nombre de vie
- Le score
- Le niveau en cours
- Le niveau de vie actuel
- Le nombre de bombes
- Le niveau de puissance (qui définit votre niveau de tir)

Les commandes sont plutôt simples :
- Les touches fléchées pour se déplacer
- La touche Espace pour tirer
- B pour utiliser une bombes

Les ennemis arrivent par le haut. Le but est d'en tuer un maximum avant de mourir et de marquer un maximum de points. A la mort d'un ennemi, des bonus peuvent apparaître.

Il y a plusieurs sortes de bonus :
- Augmente votre puissance.
- Donne une bombes.
- Redonne une vie.
- Redonne des PV.

A la fin d'un niveau, un boss apparait. A sa mort, on passe au niveau suivant.
Le jeu continue jusqu'à que le nombre de vie tombe à zéro. A la fin de la partie, on vous demande d'entrer votre nom, et vous retournez au menu principal.

B Code Source

```
#include "Bonus.h"

Bonus::Bonus(int x, int y, TypeBonus* type)
:MovableElement(x, y, 3,3, 30 * type→getScale(), 15 * type→getScale())
{my_type = type;        }

TypeBonus* Bonus::getType() const
{return my_type;        }
```



```
#include "Boss.h"
#include "constantes.h"

Boss::Boss(TypeBoss* type)
:Ennemi(type)
{
        my_type = type;
        my_changePatternDelay = type→getChangePatternDelay();
        my_actualShootPattern = type→getShootPattern();
}

ShootPattern Boss::getShootPattern() const
{return my_actualShootPattern;  }

void Boss::changePattern()
{
        if(my_changePatternClock.GetElapsedTime() > my_changePatternDelay)
        {
                switch(my_actualShootPattern)
                {
                        case SIMPLE :
                                my_actualShootPattern = M_TRIPLE;
                                my_shootingSpeed = 0.6f;
                                break;
                        case M_TRIPLE :
                                my_actualShootPattern = TOPLAYER;
                                my_shootingSpeed = 0.3f;
                                break;
                        case TOPLAYER :
                                my_actualShootPattern = HALFCIRCLE;
                                my_shootingSpeed = 0.5f;
                                break;
                        case HALFCIRCLE :
                                my_actualShootPattern = SIMPLE;
                                my_shootingSpeed = 0.2f;
                                break;
                }

                my_changePatternClock.Reset();
        }
}

void Boss::move()
{
        if(my_y < 150)
                moveDown();
        else if((my_dx < 0 ∧ my_x <0 ) ∨ (my_dx > 0 ∧ my_x + my_w > MODEL_WIDTH))
                my_dx *= -1;
        else
                my_x += my_dx;

}
```

```
#include "Ennemi.h"
#include "GameModel.h"
#include "constantes.h"
#include <cmath>
#include <iostream>

#define PI 3.14159265

using namespace std;
using namespace sf;

Ennemi::Ennemi(TypeEnnemi* type)
:Vaisseau(type→getHP(), type→getTypeTir(), type→getShootingSpeed())
{       my_type = type;
        my_dx = my_dy = type→getSpeed();
        my_w = type→getW() * ENEMY_SCALE;
        my_h = type→getH() * ENEMY_SCALE;
        my_x = rand() % (MODEL_WIDTH - my_w);
        my_y = -my_h;
                }

Ennemi::~Ennemi()
{}


int Ennemi::getScore() const
{return my_type→getScore();    }
TypeEnnemi* Ennemi::getType() const
{return my_type;        }
ShootPattern Ennemi::getShootPattern() const
{return my_type→getShootPattern();     }

void Ennemi::shoot(GameModel* model)
{
        if(my_clock.GetElapsedTime() > my_shootingSpeed )
        {
                switch(getShootPattern())
                {
                        case SIMPLE :
                        {
                                Tir* t = new Tir(my_x, my_y, 0,  my_typeTir→getSpeed(), 0, my_typeTir);
                                t→setX(my_x + my_w/2 - t→getW()/2);
                                model→addTir(t);
                        } break;

                        case M_TRIPLE :
                        {
                                Tir* T1 = new Tir(my_x, my_y, -my_typeTir→getSpeed()/3, my_typeTir→getSpeed(),0, my_typeTir); // va en bas Ã  gauche
                                Tir* T2 = new Tir(my_x, my_y, 0, my_typeTir→getSpeed(),0, my_typeTir);
//tout droit
                                Tir* T3 = new Tir(my_x, my_y, my_typeTir→getSpeed()/3, my_typeTir→getSpeed(),0, my_typeTir); //va en bas Ã  droite
                                T1→setX(my_x + my_w/2 - T1→getW()/2);
                                T2→setX(my_x + my_w/2 - T2→getW()/2);
                                T3→setX(my_x + my_w/2 - T3→getW()/2);
                                model→addTir(T1);
                                model→addTir(T2);
                                model→addTir(T3);
                        } break;

                        case TOPLAYER :
                        {
                                int DY = model→getPlayerShip()→getY() - my_y; //Distance entre l'ennemi et le joueur
                                int DX = model→getPlayerShip()→getX() - my_x;

                                float l = my_typeTir→getSpeed()/sqrt(pow(DX, 2) + pow(DY, 2)); //calcul du facteur entre la vitesse du tir et la distance entre l'ennemi et le joueur

                                float dy = l * DY; //Calcul de x et y pour que la distance parcourue par le tir soit Ã©gal Ã  celle prÃ©vue par le tir, mÃªme en diagonale.
                                float dx = l * DX;

                                Tir* t = new Tir(my_x, my_y, dx, abs(dy), 0, my_typeTir); //Prends la valeur absolue de dy, ce qui fait que le tir ne va vers le joueur que quand l'ennemi est devant lui
                             t→setX(my_x + my_w/2 - t→getW()/2);
                                model→addTir(t);
                        } break;

                        case HALFCIRCLE :
                        {
                                float rad;
                                for(int  i = 0; i ≤ 180; i += 30)
                                {
                                        rad = i * PI/180;
                                        Tir* t =  new Tir(my_x, my_y, cosf(rad) * my_typeTir→getSpeed(), sinf(rad) * my_typeTir→getSpeed(), 0, my_typeTir);
                                        t→setX(my_x + my_w/2 - t→getW()/2);
                                        model→addTir(t);
                                }
                        }break;
                }
                my_clock.Reset();
```



```
        }
}

vector<Tir*>::iterator Ennemi::isShot(GameModel* model) const
{
        vector<Tir*>::iterator shot = model→getVectorTir()→end();
    vector<Tir*>::iterator it = model→getVectorTir()→begin();
    while( it ≠ model→getVectorTir()→end() ∧ shot ≡ model→getVectorTir()→end())
    {
        if((*it)→getX() + (*it)→getW() > my_x ∧ (*it)→getX() < my_x + my_w ∧ (*it)→getY() + (*it)→getH() > my_y ∧ (*it)→getY() < my_y+my_h)
        {
                if((*it)→getPlayer())
                   shot = it;
        }
        it++;
    }

   return shot;
}

void Ennemi::move()
{ moveDown();   }
```

```
#include "GameModel.h"
#include "constantes.h"
#include <iostream>
#include <fstream>

GameModel::GameModel(int viesMax, int initialLives, int difficulty)
        :my_viesMax(viesMax), my_difficulty(difficulty)
{
        my_score = 0;

        my_tirs_types[0] = new TypeTir(4,35,20,12, "images/bullet_player1.png");
        my_tirs_types[1] = new TypeTir(4,35,40,14, "images/bullet_player2.png");
        my_tirs_types[2] = new TypeTir(4,35,60,16, "images/bullet_player3.png");

        my_tirs_types[3] = new TypeTir(14,15,20,6, "images/bullet_ennemies.png");
        my_tirs_types[4] = new TypeTir(14,15,40,8, "images/bullet_ennemies2.png");
        my_tirs_types[5] = new TypeTir(14,15,60,10, "images/bullet_ennemies3.png");
        my_tirs_types[6] = new TypeTir(14,15,80,10, "images/bullet_ennemies4.png");

        my_ennemi_types[0] = new TypeEnnemi(100, 3, 0.5f, 20, 142,115,my_tirs_types[3], SIMPLE, "images/ennemy_0.png");
        my_ennemi_types[1] = new TypeEnnemi(200, 4, 0.7f, 40, 201,179,my_tirs_types[4], M_TRIPLE, "images/ennemy_1.png");
        my_ennemi_types[2] = new TypeEnnemi(300, 5, 0.4f, 60, 155,278,my_tirs_types[5], TOPLAYER, "images/ennemy_2.png");

        my_boss_type = new TypeBoss(500, 3, 0.5f, 500, 142 * 3, 115*3, my_tirs_types[6], HALFCIRCLE, "images/ennemy_0.png",
 3);

        my_bonuses_types[0] = new TypeBonus(LIFE, 1, 0.1f, 1);
        my_bonuses_types[1] = new TypeBonus(POWER, 10, 0.5f, 1);
        my_bonuses_types[2] = new TypeBonus(BOMB, 1, 0.3f, 1);
        my_bonuses_types[3] = new TypeBonus(POWER, 100, 0.2f, 1.5f);
        my_bonuses_types[4] = new TypeBonus(HP, 80, 0.3f, 1);

        my_ship = new Joueur(0,my_tirs_types[0], initialLives, 500, 3, 0.2f);

        my_nextLevel = false;

        niveau.initNiveau(this);
        itNiveau = niveau.getMap()→begin();
}

GameModel::~GameModel()
{
        for(int i = 0; i < NB_TIRS_TYPES; i++)
                delete my_tirs_types[i];
        for(int i = 0; i < NB_ENNEMY_TYPES; i++)
                delete my_ennemi_types[i];
        for(int i = 0; i < NB_BONUS_TYPES; i++)
                delete my_bonuses_types[i];

        for(vector<Tir*>::iterator it = my_tirs.begin(); it ≠ my_tirs.end(); it++)
                delete *it;
        for(vector<Ennemi*>::iterator it = my_ennemis.begin(); it ≠ my_ennemis.end(); it++)
                delete *it;
        for(vector<Bonus*>::iterator it = my_bonuses.begin(); it ≠ my_bonuses.end(); it++)
                delete *it;

        delete my_ship;
        delete my_boss_type;
}

void GameModel::nextStep()
{

        //gestion du niveau et de l'apparition des ennemis
        if(my_clock.getTime() > itNiveau→first)
        {
                if(itNiveau ≠ niveau.getMap()→end())
                {       addEnnemy(itNiveau→second);
                        itNiveau++;     }
                else
                        if(my_ennemis.empty())
                                changeNiveau();
        }

        vector<Tir*>::iterator i; //itÃ©rateur sur vector<Tir> qui va servir Ã  plusieurs choses

        for(i = my_tirs.begin(); i ≠ my_tirs.end(); i++) //On s'en sert d'abord pour dÃ©placer tout les tirs
        {
                        (*i)→move();
                        if((*i)→getY() < 0 ∨ (*i)→getY() > MODEL_HEIGHT ∨ (*i)→getX() < 0 ∨ (*i)→getX() > MODEL_WIDTH
)
                        {       my_deletedItems.push_back(*i);
                                delete *i;
                                i = my_tirs.erase(i);
                                i--;    }
        }

        i = my_ship→isShot(this); // on se ressert de l'itÃ©rateur pour vÃ©rifier si on est touchÃ©
        if(i ≠ my_tirs.end())
        {
                my_ship→setHP(my_ship→getHP() - (*i)→getType()→getDamages());
                my_deletedItems.push_back(*i);
                delete *i;
```



```
                my_tirs.erase(i);
        }
        if(my_ship→isHitByEnemy(this) ∨ my_ship→getHP() ≤ 0)
                gameOver();

        //Puis on s'occupe des ennemis
        for(vector<Ennemi*>::iterator itE = my_ennemis.begin(); itE ≠ my_ennemis.end(); itE++)
        {
                if((*itE)→getAlive())
                {
                        Boss* b = dynamic_cast<Boss*>(*itE);
                        if(b ≠ NULL)
                                b→changePattern();

                        (*itE)→move(); //On dÃ©place l'Ã©nnemi
                        (*itE)→shoot(this); //On fait tirer l'ennemi

                        // On se ressert de l'itÃ©rateur pour cette fois vÃ©rifier si les ennemis sont touchÃ©s
                        i = (*itE)→isShot(this);
                        if( i ≠ my_tirs.end())
                        {
                                (*itE)→setHP((*itE)→getHP() - (*i)→getType()→getDamages());
                                my_deletedItems.push_back(*i);
                                delete *i;
                                my_tirs.erase(i);

                                if((*itE)→getHP() ≤0)
                                {
                                        my_score += (*itE)→getScore();
                                        addBonus((*itE)→getX(), (*itE)→getY());
                                        (*itE)→setAlive(false);
                                        itE--;
                                }
                        }
                        else if((*itE)→getY() > MODEL_HEIGHT)
                        {       my_deletedItems.push_back(*itE);
                                delete *itE;
                                my_ennemis.erase(itE);
                                itE--;
                        }
                }
        }

        //Et enfin des bonuses
        for(vector<Bonus*>::iterator itB = my_bonuses.begin(); itB ≠ my_bonuses.end(); itB++)
        {
                (*itB)→moveDown();
                if((*itB)→getX() > MODEL_HEIGHT)
                {
                        my_deletedItems.push_back(*itB);
                        my_bonuses.erase(itB);
                        itB--;
                }
                else if((*itB)→getX() + (*itB)→getW() > my_ship→getX() ∧ (*itB)→getX() < my_ship→getX()+ my_ship→get
W()
                        ∧ (*itB)→getY() + (*itB)→getH() > my_ship→getY() ∧ (*itB)→getY() < my_ship→getY() + my_ship→

getH())
                {
                        switch((*itB)→getType()→getEffect())
                        {
                                case LIFE :
                                        if(my_ship→getVies() + 1 ≤ my_viesMax)
                                                my_ship→setVies(my_ship→getVies() + (*itB)→getType()→getValue());
                                        break;
                                case BOMB : my_ship→setBombs(my_ship→getBombs() + (*itB)→getType()→getValue()); break
;
                                case POWER : my_ship→addPower((*itB)→getType()→getValue(), this); break;
                                case HP : my_ship→addHP((*itB)→getType()→getValue() ); break;
                        }

                        my_deletedItems.push_back(*itB);
                        my_bonuses.erase(itB);
                        itB--;
                }
        }
}

void GameModel::addEnnemy(TypeEnnemi* type)
{
        TypeBoss* tb = dynamic_cast<TypeBoss*>(type);
        if(tb ≠ NULL)
        {
                Boss *b = new Boss(tb);
                my_ennemis.push_back(b);
        }
        else
        {
                Ennemi *e = new Ennemi(type);
                my_ennemis.push_back(e);
        }
}

void GameModel::addTir(Tir* t)
```

```
{       my_tirs.push_back(t);   }

void GameModel::addBonus(int x, int y)
{
        //Choisit un bonus au hasard et puis fait un test pour appliquer sa proba
        int bonusType = rand() % NB_BONUS_TYPES;
        if(rand()/(float)RAND_MAX ≤ my_bonuses_types[bonusType]→getFrequency())
        {
                Bonus* b = new Bonus(x, y, my_bonuses_types[bonusType]);
                my_bonuses.push_back(b);
        }
}

void GameModel::removeEnnemy(Ennemi* ennemi)
{
        vector<Ennemi*>::iterator it = my_ennemis.begin();
        bool found = false;
        while(it ≠ my_ennemis.end() ∧ ¬found)
        {
                if(*it ≡ ennemi)
                        found = true;
                it++;
        }
        if(found)
                my_ennemis.erase(it);
}
void GameModel::removeEnnemy(vector<Ennemi*>::iterator it)
{my_ennemis.erase(it);  }

void GameModel::saveScore(string name)
{
        multimap<int, string> scores; //Utilisation d'une multimap car elle se trie toute seule et peut contenir plusieu
rs Ã©lÃ©ments qui ont la mÃªme clÃ© (ici, plusieurs personnes qui ont le mÃªme score)
        ifstream f("highScores.txt"); // Ouverture en lecture

        if(f)
        {
                int n;
                string s;

                while(f>> s ∧ f>>n)
                        scores.insert(pair<int, string>(n, s)); //On charge les scores du fichiers dans la map.

                f.close();
        }
        else
                cout << "Fichier highScores.txt inexistant, crÃ©ation d'un neuf" << endl;

        scores.insert(pair<int, string>(my_score, name)); // On ajoute le score Ã  ajouter, qui va s'insÃ©rer Ã  la bonn
e position

        ofstream f2("highScores.txt"); //Ouverture en Ã©criture

        int cpt = 0;
        if(f2)
                for(multimap<int,string>::reverse_iterator it =  scores.rbegin(); it ≠ scores.rend() ∧ cpt < 20; it++,
cpt++) //On n'enregistre que 20 scores aux maximum
                        f2 << it→second <<" " << it→first << endl; //on remet les score triÃ©s dans le fichier.
        else
                cout << "error" << endl;
}

void GameModel::changeNiveau()
{
        niveau.initNiveau(this);
        itNiveau = niveau.getMap()→begin();
        for(vector<Tir*>::iterator it = my_tirs.begin(); it ≠ my_tirs.end(); it++)
        {       my_deletedItems.push_back(*it);
                delete *it;     }
        my_tirs.clear();
        my_nextLevel = true;
        my_clock.reset();

        //RedÃ©finition du type de boss pour qu'il s'adapte au niveau en cours
        *my_boss_type = TypeBoss(500 * getLevelID(), 2 * getLevelID(), 0.5f, 500 + (200* getLevelID()), 142 * 3, 115*3,
my_tirs_types[6], HALFCIRCLE, "images/ennemy_0.png", 3);
}

void GameModel::gameOver()
{
        my_ship→setVies(my_ship→getVies() - 1);
        my_ship→setHP(my_ship→getMaxHP());
        my_ship→setX(MODEL_WIDTH/2 - my_ship→getW()/2);
        my_ship→setY(MODEL_HEIGHT - my_ship→getH());
        for(vector<Tir*>::iterator it = my_tirs.begin(); it ≠ my_tirs.end(); it++)
        {       my_deletedItems.push_back(*it);
                delete *it;     }
        my_tirs.clear();
        for(vector<Bonus*>::iterator it = my_bonuses.begin(); it ≠ my_bonuses.end(); it++)
        {       my_deletedItems.push_back(*it);
                delete *it;     }
        my_bonuses.clear();
        my_ship→setAlive(false);
```



```
}


//Accesseurs

void GameModel::getShipPos(int &x, int &y) const
{
  x = my_ship →getX();
  y = my_ship →getY();
}

bool GameModel::getNextLevel() const
{return my_nextLevel;   }
int GameModel::getLevelID() const
{return niveau.getID(); }
int GameModel::getScore() const
{return my_score;       }
int GameModel::getDifficulty() const
{return my_difficulty;  }

string GameModel::getStringLevelID() const
{
        stringstream str;
        str << niveau.getID() << endl;
        return str.str();
}

string GameModel::getStringScore() const
{
        stringstream str;
        str << my_score << endl;
        return str.str();
}

TypeEnnemi* const* GameModel::getEnnemyTypes() const
{return my_ennemi_types;        }
TypeTir* const*  GameModel::getTirTypes() const
{return my_tirs_types;  }
TypeBoss* GameModel::getBossType() const
{return my_boss_type;   }

Joueur*  GameModel::getPlayerShip() const
{       return my_ship;}


vector<Ennemi*>* GameModel::getVectorEnnemis()
{return &my_ennemis;    }
vector<Tir*>* GameModel::getVectorTir()
{return &my_tirs;       }
const vector<Bonus*> & GameModel::getVectorBonuses()
{return my_bonuses;     }
vector<MovableElement*>* GameModel::getDeletedItems()
{return &my_deletedItems;       }

Horloge*  GameModel::getClock()
{ return &my_clock;     }


void GameModel::setScore(int score)
{my_score = score;      }
void GameModel::setNextLevel(bool nextLvl)
{my_nextLevel = nextLvl;        }
```

```
#include "GameView.h"
#include "GameModel.h"
#include <iostream>
#include <fstream>
#include <sstream>

using namespace std;
using namespace sf;

GameView::GameView(int w, int h, int bpp): my_w(w), my_h(h){

  my_model = NULL;

  my_window = new RenderWindow(sf::VideoMode(w, h, bpp), "Space Sheep XTrem Turbo", sf::Style::Close);
  my_window→UseVerticalSync(true);

  my_langue = 1;
  updateLanguage();

  my_background_image = new sf::Image();
  my_background_parallax = new sf::Image();
  my_ship_image = new sf::Image();
  my_shoot_image = new sf::Image();
  my_boom_image = new sf::Image();
  my_bonus_image = new sf::Image();

  my_boom_image→LoadFromFile("images/explosion.png");

  //Initialisation des sprites
  if (¬my_background_image→LoadFromFile("images/space.png") ∨
      ¬my_background_parallax→LoadFromFile("images/stars.png") ∨
      ¬my_ship_image→LoadFromFile("images/ship_all.png")              ∨
      ¬my_bonus_image→LoadFromFile("images/bonus.png"))
  {
    my_background_sprite = GraphicElement();
    my_background_parallax_sprite = GraphicElement();
  }
  else {
    my_background_image→SetSmooth(false);
    my_background_sprite = GraphicElement(my_background_image, 0,0,w,h);
    my_background_sprite2 = GraphicElement(my_background_image,0,-h,w,h);

    my_background_parallax→SetSmooth(false);
    my_background_parallax_sprite = GraphicElement(my_background_parallax, 0, 0, w, h);
    my_background_parallax_sprite2 = GraphicElement(my_background_parallax, 0, -h, w, h);
  }

  //Initialisation des sons

  my_soundLevel = 100;
  my_soundpack = 1;

  my_titlethemeBuffer = new sf::SoundBuffer();
  my_deathmonsterBuffer = new sf::SoundBuffer();
  my_deathbossBuffer = new sf::SoundBuffer();
  my_nextLevelBuffer = new sf::SoundBuffer();
  my_gameoverBuffer = new sf::SoundBuffer();
  my_shotBuffer = new sf::SoundBuffer();

 updateSound();

}

GameView::~GameView(){
  if(my_window≠ NULL)
    delete my_window;

  delete my_background_image;
  delete my_background_parallax;
  delete my_ship_image;
  delete my_shoot_image;
  delete my_boom_image;
  delete my_bonus_image;

  delete my_titlethemeBuffer;
  delete my_deathmonsterBuffer;
  delete my_nextLevelBuffer;
  delete my_gameoverBuffer;
  delete my_shotBuffer;

  for(map<TypeEnnemi* const, sf::Image*>::iterator it = my_ennemies_images.begin(); it≠ my_ennemies_images.end(); it++)
    delete it→second;

  for(map<TypeTir* const, sf::Image*>::iterator it = my_tirs_images.begin(); it ≠ my_tirs_images.end(); it++)
    delete it→second;

  for(map<MovableElement * const, GraphicElement *>::iterator it = my_elementToGraphicElement.begin(); it ≠ my_elementTo
GraphicElement.end(); it++)
    delete it→second;
}

void GameView::setModel(GameModel * model){
```



```
  // On clear les maps synchronisÃ©e avec l'ancien modÃ¨le, le nouveau n'ayant pas forcÃ©ment les mÃªmes adresses
  if(my_model ≠ NULL)
  {
    for(map<TypeEnnemi* const, sf::Image*>::iterator it = my_ennemies_images.begin(); it≠ my_ennemies_images.end(); it++
)
      delete it→second;

    my_ennemies_images.clear();

    for(map<TypeTir* const, sf::Image*>::iterator it = my_tirs_images.begin(); it ≠ my_tirs_images.end(); it++)
      delete it→second;

    my_tirs_images.clear();

    for(map<MovableElement* const, GraphicElement*>::iterator it = my_elementToGraphicElement.begin(); it ≠ my_elementTo
GraphicElement.end(); it++)
      delete it→second;

    my_elementToGraphicElement.clear();
  }

  my_model = model;

  my_elementToGraphicElement[my_model→getPlayerShip()] = new GraphicElement (my_ship_image,0,0, my_ship_image→GetWidth
(),my_ship_image→GetHeight());
  my_elementToGraphicElement[my_model→getPlayerShip()]→SetScale(SHEEP_SCALE, SHEEP_SCALE);
  my_elementToGraphicElement[my_model→getPlayerShip()]→SetSubRect(sf::IntRect(0,0,SHEEP_WIDTH, SHEEP_HEIGHT));

  for(int i = 0; i < NB_ENNEMY_TYPES; i++)
  {
     my_ennemies_images[my_model→getEnnemyTypes()[i]] = new sf::Image();
     my_ennemies_images[my_model→getEnnemyTypes()[i]]→LoadFromFile(my_model→getEnnemyTypes()[i]→getFilename());
  }
  my_ennemies_images[my_model→getBossType()] = new sf::Image();
  my_ennemies_images[my_model→getBossType()]→LoadFromFile(my_model→getBossType()→getFilename());

  for(int i = 0; i < NB_TIRS_TYPES; i++)
  {
    my_tirs_images[my_model→getTirTypes()[i]] = new sf::Image();
    my_tirs_images[my_model→getTirTypes()[i]]→LoadFromFile(my_model→getTirTypes()[i]→getFilename());
  }
}

void GameView::updateLanguage()
{
  string langue;
  switch(my_langue)
  {
    case 1 : langue = "French"; break;
    case 2 : langue = "English"; break;
  }

  string filename = "LanguageFiles/" + langue + ".txt";
  ifstream f(filename.c_str());

  if(f)
  {
    string s;

    for(int i = 0; i < NB_STRINGS; i++)
      getline(f,my_strings[i]);

  }
}

void GameView::updateSound()
{
  string soundpack, extension;
  switch(my_soundpack)
  {
    case 1 : soundpack = "8bit"; extension = ".wav" ;break;
    case 2 : soundpack = "Human"; extension = ".ogg" ;break;
  }

  string directoryPath = "Music/" + soundpack + "/" ;

  *my_titlethemeBuffer =  sf::SoundBuffer();
  *my_deathmonsterBuffer =  sf::SoundBuffer();
  *my_deathbossBuffer =  sf::SoundBuffer();
  *my_nextLevelBuffer =  sf::SoundBuffer();
  *my_gameoverBuffer =  sf::SoundBuffer();
  *my_shotBuffer =  sf::SoundBuffer();

  if(¬my_titlethemeBuffer→LoadFromFile(directoryPath + "title" + extension) ∨
  ¬my_maintheme.OpenFromFile(directoryPath + "maintheme" + extension) ∨
  ¬my_deathmonsterBuffer→LoadFromFile(directoryPath + "deathmonster" + extension) ∨
  ¬my_deathbossBuffer→LoadFromFile(directoryPath + "deathboss" + extension) ∨
  ¬my_nextLevelBuffer→LoadFromFile(directoryPath + "nextlevel" + extension) ∨
  ¬my_gameoverBuffer→LoadFromFile(directoryPath + "gameover" + extension) ∨
  ¬my_shotBuffer→LoadFromFile(directoryPath + "shot" + extension) )
  {
    my_titletheme = Sound();
    my_deathmonster = Sound();
```

```
    my_deathboss = Sound();
    my_nextLevel = Sound();
    my_gameover = Sound();
    my_shotsound =  Sound();
  }
  else
  {
    my_maintheme.SetLoop(true);
    my_titletheme = Sound(*my_titlethemeBuffer);
    my_nextLevel = Sound(*my_nextLevelBuffer);
    my_deathmonster = Sound(*my_deathmonsterBuffer);
    my_deathboss = Sound(*my_deathbossBuffer);
    my_gameover = Sound(*my_gameoverBuffer);
    my_shotsound = Sound(*my_shotBuffer);
  }

}

void GameView::updateVolume()
{
  my_titletheme.SetVolume(my_soundLevel);
  my_maintheme.SetVolume(my_soundLevel);
  my_nextLevel.SetVolume(my_soundLevel);
  my_deathmonster.SetVolume(my_soundLevel);
  my_gameover.SetVolume(my_soundLevel);
  my_shotsound.SetVolume(my_soundLevel);
}

void GameView::bgscroll()
{
  my_background_sprite.Move(0,1);
  my_background_sprite2.Move(0,1);
  if(my_background_sprite.GetPosition().y ≡ VIEW_HEIGHT)
    my_background_sprite.SetPosition(0,-my_h);
  if(my_background_sprite2.GetPosition().y ≡ VIEW_HEIGHT)
    my_background_sprite2.SetPosition(0,-my_h);

  my_background_parallax_sprite.Move(0, 2);
  my_background_parallax_sprite2.Move(0, 2);
  if(my_background_parallax_sprite.GetPosition().y ≡ VIEW_HEIGHT)
      my_background_parallax_sprite.SetPosition(0, -my_h);
  if(my_background_parallax_sprite2.GetPosition().y ≡ VIEW_HEIGHT)
      my_background_parallax_sprite2.SetPosition(0, -my_h);
}

void GameView::explosion(GraphicElement &elem, Ennemi* adr)
{
  if(elem.GetImage() ≠ my_boom_image)
  {
    elem = GraphicElement(my_boom_image, elem.GetPosition().x,elem.GetPosition().y, my_boom_image→GetWidth(), my_boom_i
mage→GetHeight());
    elem.SetSubRect(sf::IntRect(0,0, 200,205));

    Boss* b = dynamic_cast<Boss*>(adr);
    if(b ≠ NULL)
      my_deathboss.Play();
    else
      my_deathmonster.Play();
  }
  else
    elem.SetSubRect(sf::IntRect(elem.GetSubRect().Left + 200, 0, elem.GetSubRect().Right + 200, 205));

}

void GameView::draw()
{
  if(my_titletheme.GetStatus() ≠ sf::Sound::Stopped)
    my_titletheme.Stop();
  if(my_maintheme.GetStatus() ≠ sf::Sound::Playing)
    my_maintheme.Play();

  my_window→Clear();

  if(my_model→getPlayerShip()→getAlive()) //Si vivant
  {
    if(¬my_model→getNextLevel())
    {
      bgscroll();
      my_window→Draw(my_background_sprite);
      my_window→Draw(my_background_sprite2);
      my_window→Draw(my_background_parallax_sprite);
      my_window→Draw(my_background_parallax_sprite2);

      for(map<MovableElement * const, GraphicElement *>::iterator it = my_elementToGraphicElement.begin(); it≠my_element
ToGraphicElement.end();it++)
        if(it→second→getVisible())
          it→second→draw(my_window);

      my_elementToGraphicElement[my_model→getPlayerShip()]→draw(my_window);
      my_hud.drawHUD(my_window);

    }
    else
```



```
    {
      my_maintheme.Pause();
      my_nextLevel.Play();
      transition(my_strings[FINISHLEVEL], 3);

      my_model→setNextLevel(false);
    }
  }
  else
  {
    if(my_model→getPlayerShip()→getVies() ≡ 0 ) //Si plus de vies->transition game over
    {
      my_maintheme.Pause();
      my_gameover.Play();
      transition(my_strings[GAMEOVER], 3 );

      askName();
    }
    else // Si il reste encore des vies -> Ã©cran de transition de perte de vie
    {
      transition(my_strings[DEAD], 3 );

      my_model→getPlayerShip()→setAlive(true);
    }
  }
}

void GameView::Display()
{
  my_window→Display();
}

void GameView::transition(string text, float t)
{
  my_model→getClock()→pause();

  sf::String sText = String(text);
  sText.SetFont(sf::Font::GetDefaultFont());
  sText.SetPosition(VIEW_WIDTH/2 - sText.GetRect().GetWidth()/2,VIEW_HEIGHT/2 - sText.GetRect().GetHeight()/2);
  sText.SetColor(Color(255,255,255));

  sf::String sCount = String();

  sf::Clock countdown;

  while(countdown.GetElapsedTime() < t)
  {

    int IntCountDown = countdown.GetElapsedTime() ;
    my_window→Clear();

    stringstream sstr;
    sstr << t - IntCountDown << endl;

    sCount.SetText(sstr.str());
    sCount.SetPosition(VIEW_WIDTH/2 - sCount.GetRect().GetWidth()/2, VIEW_HEIGHT - 150);

    my_window→Draw(sText);
    my_window→Draw(sCount);
    my_window→Display();
  }

  my_model→getClock()→start();
}

void GameView::synchronize()
{
  //synchro du joueur
        int x_ship, y_ship;
  my_model→getShipPos(x_ship, y_ship);
  my_elementToGraphicElement[my_model→getPlayerShip()]→SetPosition(x_ship, y_ship);

  //update du HUD
  my_hud.update(my_model, my_strings);

  //synchro des tirs
  for(vector<Tir*>::iterator it = my_model→getVectorTir()→begin(); it ≠ my_model→getVectorTir()→end(); it++)
  {
    if(my_elementToGraphicElement.count(*it) ≡ 0)
      my_elementToGraphicElement[*it] = new GraphicElement();

    my_elementToGraphicElement[*it]→setVisible(true);

    /*Vu qu'on dÃ©truit les Ã©lements de la map que dans le destrcteur, si dans le modÃ¨le quand on crÃ©e un Ã©lement,
     il prend une adresse dÃ©jÃ  rÃ©fÃ©rencÃ©e, la map l'associe au GraphicElement dÃ©jÃ  existant(qui peut Ãªtre n'impo
rte quoi),
     du coup faut redÃ©finir le GraphicElement */
    *(my_elementToGraphicElement[*it]) = GraphicElement(my_tirs_images[(*it)→getType()], 0,0,my_tirs_images[(*it)→getT
ype()]→GetWidth(), my_tirs_images[(*it)→getType()]→GetHeight());
    my_elementToGraphicElement[*it]→resize((*it)→getW(), (*it)→getH());
    my_elementToGraphicElement[*it]→SetPosition((*it)→getX(), (*it)→getY());

    if(¬(*it)→getPlayer())
```

```
    {
      my_elementToGraphicElement[*it]→SetRotation(180);
      my_elementToGraphicElement[*it]→Move(my_elementToGraphicElement[*it]→GetSize().x, my_elementToGraphicElement[*it
]→GetSize().y);
    }
    else
      my_elementToGraphicElement[*it]→SetRotation(0);
  }


  //synchro des ennemis
  for(vector<Ennemi*>::iterator itE = my_model→getVectorEnnemis()→begin(); itE≠ my_model→getVectorEnnemis()→end(); it
E++)
  {
    if(my_elementToGraphicElement.count(*itE) ≡ 0)
      my_elementToGraphicElement[*itE] = new GraphicElement();
    my_elementToGraphicElement[*itE]→setVisible(true);

    Boss* b = dynamic_cast<Boss*>(*itE);
    if(b ≠ NULL)
      my_hud.updateBossPV(b);

    if((*itE)→getAlive())
    {
      //Idem qu'au dessus
      *(my_elementToGraphicElement[*itE]) = GraphicElement(my_ennemies_images[(*itE)→getType()], 0, 0 ,my_ennemies_imag
es[(*itE)→getType()]→GetWidth(), my_ennemies_images[(*itE)→getType()]→GetHeight());

      my_elementToGraphicElement[*itE]→resize((*itE)→getW(), (*itE)→getH());
      my_elementToGraphicElement[*itE]→SetPosition((*itE)→getX(), (*itE)→getY());
    }
    else
    {
      if(my_elementToGraphicElement[*itE]→GetSubRect().Right ≡ 600) // Si l'explosion est finie, on supprime
      { my_model→getDeletedItems()→push_back(*itE);
        delete *itE;
        my_model→removeEnnemy(itE);
        itE--;  }
      else // Sinon, on la continue
      {
        if((*itE)→getClock()→GetElapsedTime() > 0.2)
        {
          explosion(*(my_elementToGraphicElement[*itE]), *itE);
          my_elementToGraphicElement[*itE]→SetScale(0.5, 0.5);
          my_elementToGraphicElement[*itE]→SetPosition((*itE)→getX() - 50 + (*itE)→getW()/2, (*itE)→getY() - 51 + (*i
tE)→getH()/2);
          (*itE)→getClock()→Reset();
        }
      }
    }
  }


  //synchro des Bonus
  for(vector<Bonus*>::const_iterator itB = my_model→getVectorBonuses().begin(); itB ≠ my_model→getVectorBonuses().end(
); itB++)
  {
    if(my_elementToGraphicElement.count(*itB) ≡ 0)
      my_elementToGraphicElement[*itB] = new GraphicElement();

    my_elementToGraphicElement[*itB]→setVisible(true);

    *(my_elementToGraphicElement[*itB]) = GraphicElement(my_bonus_image, 0, 0, my_bonus_image→GetWidth(), my_bonus_imag
e→GetHeight());
    my_elementToGraphicElement[*itB]→SetPosition((*itB)→getX(), (*itB)→getY());
    my_elementToGraphicElement[*itB]→resize((*itB)→getW(), (*itB)→getH());
    switch((*itB)→getType()→getEffect())
    {
      case LIFE : my_elementToGraphicElement[*itB]→SetColor(sf::Color::Green); break;
      case POWER: my_elementToGraphicElement[*itB]→SetColor(sf::Color::Red); break;
      case BOMB : my_elementToGraphicElement[*itB]→SetColor(sf::Color::Blue); break;
      case HP : my_elementToGraphicElement[*itB]→SetColor(Color(255,20,147)); break;
    }
  }

  //On rend invisible les GraphicElements de la map dont la key n'est ni l'adresse d'un ennemi, ni celle d'un tir, ni ce
lle du joueur

  for(vector<MovableElement*>::iterator it = my_model→getDeletedItems()→begin(); it ≠ my_model→getDeletedItems()→end(
); it++)
  {
    /*Met seulement invisible, pour Ã©viter d'avoir Ã  faire des new et delete, et permettre de rÃ©cupÃ©rer le GraphicEl
ement si l'adresse est rÃ©allouÃ©e,
      mais provoque des erreurs de segmentations par moment, pour une raison inconnue */
    // my_elementToGraphicElement[*it]->setVisible(false);


    /* Moins efficace que la mÃ©thode du dessus car fait un delete sur chaque Ã©lÃ©ment dÃ©truit, et donc provoquera un
new si l'adresse est rÃ©allouÃ©e,
      mais ne provoque pas d'erreur de segmentation */
     delete my_elementToGraphicElement[*it];
     my_elementToGraphicElement.erase(*it);
  }
```



```
  my_model→getDeletedItems()→clear();
}


bool GameView::treatEvents(bool &quit){
  bool result = false;
  const sf::Input& Input = my_window→GetInput();

  if(my_window→IsOpened())
  {
    result = true;
    Event event;
    while (my_window→GetEvent(event))
    {
              if (event.Type ≡ Event::Closed)
    {
                      my_window→Close();
                      result = false;
      quit = true;
                    }
      if((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Escape))
      {
        my_model→getClock()→pause();
        switch(pause())
        {
          case 1 : my_model→getClock()→start(); break;
          case 2 : my_maintheme.Pause();
                   result = false;
                   break;
          case 3 : my_window→Close();
                   result = false;
                   quit = true;
                   break;
        }
      }
    }

    my_elementToGraphicElement[my_model→getPlayerShip()]→SetSubRect(sf::IntRect(0,0,SHEEP_WIDTH, SHEEP_HEIGHT));

    if(Input.IsKeyDown(sf::Key::Left))
    { my_model→getPlayerShip()→moveLeft();
      my_elementToGraphicElement[my_model→getPlayerShip()]→SetSubRect(sf::IntRect(200,0,400, SHEEP_HEIGHT)); }
        if(Input.IsKeyDown(sf::Key::Right))
        {       my_model→getPlayerShip()→moveRight();
      my_elementToGraphicElement[my_model→getPlayerShip()]→SetSubRect(sf::IntRect(400,0,600, SHEEP_HEIGHT)); }
        if(Input.IsKeyDown(sf::Key::Up))
                my_model→getPlayerShip()→moveUp();
        if(Input.IsKeyDown(sf::Key::Down))
                my_model→getPlayerShip()→moveDown();

        if(my_clock.GetElapsedTime() > 0.2)
        {
                if(Input.IsKeyDown(sf::Key::Space))
                {       my_model→getPlayerShip()→shoot(my_model);
      my_shotsound.Play();
      my_clock.Reset();}
      if(Input.IsKeyDown(sf::Key::B))
        {my_model→getPlayerShip()→useBomb(my_model);
          my_clock.Reset(); }

      if(Input.IsKeyDown(sf::Key::E))
      {  my_model→addEnnemy(my_model→getEnnemyTypes()[1]);
        my_clock.Reset(); }
      if(Input.IsKeyDown(sf::Key::Z))
       { my_model→addEnnemy(my_model→getEnnemyTypes()[0]);
         my_clock.Reset(); }
      if(Input.IsKeyDown(sf::Key::R))
       { my_model→addEnnemy(my_model→getEnnemyTypes()[2]);
        my_clock.Reset(); }
        }


    if(my_model→getPlayerShip()→getVies() ≡ 0 )
    {
      result = false;
    }
  }
  return result;
}



int GameView::pause()
{
  sf::String sPause = String(my_strings[PAUSE]);
  sPause.SetPosition(VIEW_WIDTH/2 - sPause.GetRect().GetWidth()/2, 150);

  sf::String sResume = String(my_strings[RESUME]);
  sResume.SetPosition(VIEW_WIDTH/2 - sResume.GetRect().GetWidth()/2, 250);
```

```
  sf::String sBack = String(my_strings[BACKTOMENU]);
  sBack.SetPosition(VIEW_WIDTH/2 - sBack.GetRect().GetWidth()/2, 300);

  sf::String sQuit = String(my_strings[QUIT]);
  sQuit.SetPosition(VIEW_WIDTH/2 - sQuit.GetRect().GetWidth()/2, 350);

  sf::Shape rectPause = sf::Shape::Rectangle(100, 100, VIEW_WIDTH - 100, VIEW_HEIGHT - 100, sf::Color::Black, 3, sf::Col
or::White);

  int result = 0;
  int choice = 1;

  if(my_window->IsOpened())
  {
    Event event;

    while(result == 0)
    {
      my_window->Clear();
      draw();

      sResume.SetColor(sf::Color::White);
      sBack.SetColor(sf::Color::White);
      sQuit.SetColor(sf::Color::White);

      switch(choice)
      {
        case 1 : sResume.SetColor(sf::Color::Red); break;
        case 2 : sBack.SetColor(sf::Color::Red); break;
        case 3 : sQuit.SetColor(sf::Color::Red); break;
      }

      my_window->Draw(rectPause);
      my_window->Draw(sPause);
      my_window->Draw(sResume);
      my_window->Draw(sBack);
      my_window->Draw(sQuit);
      my_window->Display();

      while(my_window->GetEvent(event))
      {
        if (event.Type == Event::Closed)
        {
          result = 3;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Up))
        {
          if(choice - 1 > 0)
            choice--;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Down))
        {
          if(choice + 1 <= 3)
            choice++;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Return))
          result = choice;
      }
    }
  }
  return result;
}


void GameView::intro()
{
  sf::String names = String(L"HarscoÃ«t-Chassaigne Productions");
  names.SetFont(sf::Font::GetDefaultFont());
  names.SetPosition(VIEW_WIDTH/2 - names.GetRect().GetWidth()/2, VIEW_HEIGHT/2 - names.GetRect().GetHeight());

  sf::String presents = String("Presents");
  presents.SetFont(sf::Font::GetDefaultFont());
  presents.SetPosition(VIEW_WIDTH/2 - presents.GetRect().GetWidth()/2, VIEW_HEIGHT/2);

  int alpha = 0;
  for(alpha = 0; alpha <= 255; alpha+= 2)
  {
    my_window->Clear();
    names.SetColor(Color(255,255,255,alpha));
    presents.SetColor(Color(255,255,255,alpha));
    my_window->Draw(names);
    my_window->Draw(presents);
    my_window->Display();
  }
  for(alpha = 255; alpha >= 0; alpha-=2)
  {
    my_window->Clear();
    names.SetColor(Color(255,255,255,alpha));
    presents.SetColor(Color(255,255,255,alpha));
    my_window->Draw(names);
    my_window->Draw(presents);
    my_window->Display();
  }
```



```
}

int GameView::menu()
{

  sf::Font shmupFont;
  shmupFont.LoadFromFile("Fonts/Spaceman.ttf");

  sf::String shmup = String("Space Sheep Xtrem Turbo \n      Penguin Edition");
  shmup.SetFont(shmupFont);
  shmup.SetSize(25);
  shmup.SetPosition(VIEW_WIDTH/2 - shmup.GetRect().GetWidth()/2, 50);
  shmup.SetColor(Color(0,0,0));

  sf::String play = String(my_strings[JOUER]);
  play.SetFont(sf::Font::GetDefaultFont());
  play.SetPosition(50,200);

  sf::String highscores = String(my_strings[HIGHSCORES]);
  highscores.SetPosition(50, 300);

  sf::String settings = String(my_strings[SETTINGS]);
  settings.SetPosition(50, 400);

  sf::String quit = String(my_strings[QUIT]);
  quit.SetPosition(50,500);

  if(my_titletheme.GetStatus() != sf::Sound::Playing)
    my_titletheme.Play();


  int result = 0;
  int choice = 1;

  if(my_window->IsOpened())
  {
    Event event;

    while(result == 0)
    {
      my_window->Clear(Color(255,255,255));
      my_window->Draw(shmup);

      play.SetColor(sf::Color::Black);
      highscores.SetColor(sf::Color::Black);
      settings.SetColor(sf::Color::Black);
      quit.SetColor(sf::Color::Black);

      switch(choice)
      {
        case 1 : play.SetColor(sf::Color::Red); break;
        case 2 : highscores.SetColor(sf::Color::Red); break;
        case 3 : settings.SetColor(sf::Color::Red); break;
        case 4 : quit.SetColor(sf::Color::Red); break;
      }

      my_window->Draw(play);
      my_window->Draw(highscores);
      my_window->Draw(settings);
      my_window->Draw(quit);
      my_window->Display();

      while(my_window->GetEvent(event))
      {
        if ((event.Type == Event::Closed) ||
        ((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Escape)))
        {
          my_window->Close();
          result = 4;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Up))
        {
          if(choice - 1 > 0)
            choice--;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Down))
        {
          if(choice + 1 <= 4)
            choice++;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Return))
          result = choice;
      }
    }
  }
  return result;
}

void GameView::options(bool &quit, int &difficulty, int &initialLives)
{
  if(my_window->IsOpened())
  {
```

```
    sf::String sSoundLevel = String();
    sSoundLevel.SetPosition(50, 100);

    sf::String sDifficulty = String();
    sDifficulty.SetPosition(50, 150);

    sf::String sLives = String();
    sLives.SetPosition(50, 200);

    sf::String sLanguage = String();
    sLanguage.SetPosition(50, 250);

    sf::String sSoundPack = String();
    sSoundPack.SetPosition(50, 300);

    sf::String sDifficultyLevel = String();
    sDifficultyLevel.SetPosition(500, 150);
    sDifficultyLevel.SetColor(sf::Color::Black);

    sf::String sMyLanguage = String();
    sMyLanguage.SetPosition(500, 250);
    sMyLanguage.SetColor(sf::Color::Black);

    sf::String sSelectedSoundPack = String();
    sSelectedSoundPack.SetPosition(500, 300);
    sSelectedSoundPack.SetColor(sf::Color::Black);

    sf::Shape sLifeCircle = sf::Shape::Circle(0,0, 10, sf::Color::White);
    sf::Shape sSoundRect;
    sf::Shape sSoundRectFond = sf::Shape::Rectangle(500, sSoundLevel.GetPosition().y +10, 700,sSoundLevel.GetPosition().
y + 20, Color(188,188,188), 3, Color(125,125,125));

    Event event;
    bool backToMenu = false;

    int choice = 0;

    while(¬backToMenu)
    {
      my_window→Clear(sf::Color::White);
      updateVolume();
      updateLanguage();
      updateSound();

      sSoundLevel.SetText(my_strings[SOUNDLEVEL] + ":");
      sDifficulty.SetText(my_strings[DIFFICULTY] + ":");
      sLives.SetText(my_strings[LIVESNUMBER] + ":");
      sLanguage.SetText(my_strings[LANGUAGE] + ":");
      sSoundPack.SetText(my_strings[SOUNDPACK] + ":" );

      sSoundLevel.SetColor(sf::Color::Black);
      sDifficulty.SetColor(sf::Color::Black);
      sLives.SetColor(sf::Color::Black);
      sLanguage.SetColor(sf::Color::Black);
      sSoundPack.SetColor(sf::Color::Black);

      switch(choice)
      {
        case 0 : sSoundLevel.SetColor(sf::Color::Red); break;
        case 1 : sDifficulty.SetColor(sf::Color::Red); break;
        case 2 : sLives.SetColor(sf::Color::Red);break;
        case 3 : sLanguage.SetColor(sf::Color::Red); break;
        case 4 : sSoundPack.SetColor(sf::Color::Red); break;
      }

      my_window→Draw(sSoundLevel);
      my_window→Draw(sDifficulty);
      my_window→Draw(sLives);
      my_window→Draw(sLanguage);
      my_window→Draw(sSoundPack);


      sSoundRect = sf::Shape::Rectangle(500, sSoundLevel.GetPosition().y + 10, 500 + 2 * my_soundLevel, sSoundLevel.GetP
osition().y + 20, sf::Color::Black);
      my_window→Draw(sSoundRectFond);
      my_window→Draw(sSoundRect);

      sLifeCircle.SetColor(sf::Color::Black);
      for(int i=0; i < initialLives; i++)
      {
        sLifeCircle.SetPosition( 500 + 30*i, 215);
        my_window→Draw(sLifeCircle);
      }

      switch(difficulty)
      {
        case 1 : sDifficultyLevel.SetText(my_strings[NORMAL]); break;
        case 2 : sDifficultyLevel.SetText(my_strings[HARD]); break;
        case 3 : sDifficultyLevel.SetText(my_strings[VERYHARD]); break;
      }

      my_window→Draw(sDifficultyLevel);
```



```
      sMyLanguage.SetText(my_strings[MYLANGUE]);
      my_window→Draw(sMyLanguage);

      switch(my_soundpack)
      {
        case 1 : sSelectedSoundPack.SetText("8-Bit"); break;
        case 2 : sSelectedSoundPack.SetText("Human Mouth"); break;
      }

      my_window→Draw(sSelectedSoundPack);

      while(my_window→GetEvent(event))
      {
        if ((event.Type ≡ Event::Closed) ∨
          ((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Escape)))
        {
          my_window→Close();
          backToMenu = true;
          quit = true;
        }

        if((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Up))
          if(choice > 0)
            choice--;

        if((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Down))
          if(choice < 4)
            choice++;

        if((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Right))
        {
          switch(choice)
          {
            case 0 : if(my_soundLevel < 100) my_soundLevel += 20; break;
            case 1 : if(difficulty < 3) difficulty++; break;
            case 2 : if(initialLives < 5) initialLives++; break;
            case 3 : if(my_langue < 2) my_langue++; break;
            case 4 : if(my_soundpack < 2) my_soundpack++; break;
          }
        }

        if((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Left))
        {
          switch(choice)
          {
            case 0 : if(my_soundLevel > 0) my_soundLevel -= 20; break;
            case 1 : if(difficulty > 1) difficulty--; break;
            case 2 : if(initialLives > 1) initialLives--; break;
            case 3 : if(my_langue > 1) my_langue--; break;
            case 4 : if(my_soundpack > 1 ) my_soundpack-- ; break;
          }
        }


        if((event.Type ≡ Event::KeyPressed) ∧ (event.Key.Code ≡ sf::Key::Return))
          backToMenu = true;
      }

      my_window→Display();
    }
  }
}

void GameView::viewHighscores(bool & quit)
{
  if(my_window→IsOpened())
  {
    my_window→Clear(sf::Color::White);

    sf::String sfString;
    sfString.SetColor(sf::Color::Black);

    ifstream f("highScores.txt");
    if(f)
    {
      string score, name;

      int y = 0;

      while(f >> name ∧ f >> score)
      {
        y += 30;

        sfString.SetText(name);
        sfString.SetPosition(50,y);
        my_window→Draw(sfString);

        sfString.SetText(score);
        sfString.SetPosition(400,y);
        my_window→Draw(sfString);
      }
    }
    else
```

```
    {
      sfString.SetText(my_strings[NOSCORE]);
      sfString.SetPosition(VIEW_WIDTH/2 - sfString.GetRect().GetWidth()/2,VIEW_HEIGHT/2 - sfString.GetRect().GetHeight()
/2);
      my_window->Draw(sfString);
    }

    my_window->Display();

    Event event;
    bool backToMenu = false;
    while(my_window->GetEvent(event) || !backToMenu)
    {
        if(  /*(event.Type == Event::Closed) ||*/ /* < (BUG :) Cette partie de la condition fait quitter le jeu Ã  l'aff
ichage des scores, pour une raison inconnue. (On ne peut du coup pas fermer la fenÃªtre de faÃ§on "normale" si l'on affi
che les "High Scores". */
           ((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Escape)))
        {
          my_window->Close();
          backToMenu = true;
          quit = true;
        }
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Return))
          backToMenu = true;
    }
  }
}

void GameView::askName()
{
  if(my_window->IsOpened())
  {
    sf::String s = String();

    string name;

    Event event;
    bool finished = false;

    while(!finished)
    {
      my_window->Clear();

      s.SetText(my_strings[ENTERNAME] + ":");
      s.SetPosition(200, 200);
      s.SetColor(sf::Color::White);
      my_window->Draw(s);

      while(my_window->GetEvent(event))
      {
        if(event.Type == Event::TextEntered && name.size() < 16
          && event.Text.Unicode > 30 && (event.Text.Unicode < 127 || event.Text.Unicode > 159) && event.Text.Unicode != 32)
            name += event.Text.Unicode;

        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Back) && name.size() > 0)
          name.erase(name.size()-1);
        if((event.Type == Event::KeyPressed) && (event.Key.Code == sf::Key::Return))
          finished = true;
      }

      s.SetText(name);
      s.SetPosition(300,250);
      my_window->Draw(s);

      my_window->Display();
    }

    my_model->saveScore(name);
  }
}
```



```
#include "GraphicElement.h"
#include <iostream>


using namespace std;
using namespace sf;

GraphicElement::GraphicElement():  Sprite(), _w(0), _h(0)
{}

GraphicElement::GraphicElement(Image * image, int x, int y, int w, int h)
  :  Sprite(*image), _w(w), _h(h)
{
  Resize(w, h);
  SetPosition(x,y);
  _visible = true;
}

GraphicElement::~GraphicElement()
{}

void GraphicElement::setPosition(int x, int y){
  SetPosition(x, y);
}

void GraphicElement::setPosition(const Vector2f & pos){
  SetPosition(pos.x, pos.y);
}

void GraphicElement::resize(int w, int h){
  _w = w;
  _h = h;
  Resize(_w, _h);
}

bool GraphicElement::getVisible() const
{return _visible;       }
void GraphicElement::setVisible(bool visible)
{_visible = visible;    }

void GraphicElement::draw(RenderWindow * window){
  window->Draw(*this);
}
```

```
#include "Horloge.h"

Horloge::Horloge()
{       clock_gettime(CLOCK_REALTIME, &tp);
        m_startTime  = tp.tv_sec;
        m_paused = false;
        m_buffer = 0;}

float Horloge::getTime()
{       if(m_paused)
                return m_buffer;
        else
        {
                clock_gettime(CLOCK_REALTIME, &tp);
                return (float)(tp.tv_sec - m_startTime) + m_buffer;
        }
}

bool Horloge::getStatus() const
{return m_paused;       }


void Horloge::pause()
{
        if(!m_paused)
        {
                m_buffer = getTime();
                m_paused = true;
        }
}

void Horloge::start()
{
        if(m_paused)
        {
                clock_gettime(CLOCK_REALTIME, &tp);
                m_startTime  = tp.tv_sec;
                m_paused = false;
        }
}

void Horloge::stop()
{
        pause();
        reset();
}

void Horloge::reset()
{
        clock_gettime(CLOCK_REALTIME, &tp);
        m_startTime  = tp.tv_sec;
        m_buffer = 0;
}
```



```
#include "Interface.h"
#include "constantes.h"
#include <fstream>
#include <iostream>

using namespace std;
using namespace sf;

Interface::Interface()
{
        my_HUDsfStrings[VIES].SetFont(my_font.GetDefaultFont());
        my_HUDsfStrings[VIES].SetPosition(0,0);
        my_HUDsfStrings[VIES].SetColor(sf::Color::Black);

          my_HUDsfStrings[PV].SetFont(my_font.GetDefaultFont());
          my_HUDsfStrings[PV].SetPosition(0,30);
          my_HUDsfStrings[PV].SetColor(sf::Color::Black);

          my_HUDsfStrings[NIVEAU].SetFont(my_font.GetDefaultFont());
          my_HUDsfStrings[NIVEAU].SetPosition(VIEW_WIDTH - 150 ,0);
        my_HUDsfStrings[NIVEAU].SetColor(sf::Color::Black);

        my_HUDsfStrings[BOMBES].SetFont(my_font.GetDefaultFont());
        my_HUDsfStrings[BOMBES].SetPosition(0, 60);
        my_HUDsfStrings[BOMBES].SetColor(sf::Color::Black);

        my_HUDsfStrings[PUISSANCE].SetFont(my_font.GetDefaultFont());
        my_HUDsfStrings[PUISSANCE].SetPosition(VIEW_WIDTH - 250, 60);
        my_HUDsfStrings[PUISSANCE].SetColor(sf::Color::Black);

        my_HUDsfStrings[5].SetFont(my_font.GetDefaultFont());
        my_HUDsfStrings[5].SetPosition(VIEW_WIDTH/2 - my_HUDsfStrings[5].GetRect().GetWidth()/2,0);
        my_HUDsfStrings[5].SetColor(sf::Color::Black);

        my_revive_image = new sf::Image();
        if(!my_revive_image->LoadFromFile("images/revive.png"))
                my_revive_sprite_template = GraphicElement();
        else
        {
                my_revive_sprite_template = GraphicElement(my_revive_image,0,0,my_revive_image->GetWidth(), my_revive_im
age->GetHeight());
            my_revive_sprite_template.SetScale(0.1,0.1);
    }
  }

Interface::~Interface()
{
        delete my_revive_image;
}

void Interface::drawHUD(sf::RenderWindow * _window)
{
        for(int i = 0; i < 6; i++)
                _window->Draw(my_HUDsfStrings[i]);

        for(int i = 0; i < my_nbVies; i++)
        {
         my_revive_sprite_template.SetPosition(90 + 30*i, 10);
         my_revive_sprite_template.draw(_window);
        }

        _window->Draw(my_rectPVFond);
  _window->Draw(my_rectPV);
  _window->Draw(my_rectBossPV);
}


void Interface::update(GameModel* model, string strings[])
{
        my_HUDsfStrings[5].SetText(model->getStringScore());
        my_HUDsfStrings[5].SetPosition(VIEW_WIDTH/2 - my_HUDsfStrings[5].GetRect().GetWidth()/2,0);

        my_HUDsfStrings[VIES].SetText(strings[VIES] + ":");
        my_HUDsfStrings[PV].SetText(strings[PV] + ":" );
        my_HUDsfStrings[NIVEAU].SetText(strings[NIVEAU] + ":" + model->getStringLevelID());

        my_HUDsfStrings[BOMBES].SetText(strings[BOMBES] + ":" + model->getPlayerShip()->getStringBombs());

        my_HUDsfStrings[PUISSANCE].SetText(strings[PUISSANCE] + ":" + model->getPlayerShip()->getStringPower());
        my_HUDsfStrings[PUISSANCE].SetPosition(VIEW_WIDTH - my_HUDsfStrings[PUISSANCE].GetRect().GetWidth(), 60);

        my_nbVies = model->getPlayerShip()->getVies();

        my_rectPVFond = sf::Shape::Rectangle(70,48,70 + model->getPlayerShip()->getMaxHP()/2, 58, Color(188,188,188), 3,
 Color(125,125,125));

        my_rectPV = sf::Shape::Rectangle(70, 48, 70+ model->getPlayerShip()->getHP()/2, 58, sf::Color::White);
        my_rectPV.SetColor(sf::Color::Green);
        if(model->getPlayerShip()->getHP() <= 50 * (model->getPlayerShip()->getMaxHP()/100))
        my_rectPV.SetColor(sf::Color::Yellow);
        if(model->getPlayerShip()->getHP() <= 20 * (model->getPlayerShip()->getMaxHP()/100))
        my_rectPV.SetColor(sf::Color::Red);
```

```
  my_rectBossPV = sf::Shape();

}


void Interface::updateBossPV(Boss* boss)
{
  if(boss→getHP() > 0)
    my_rectBossPV = sf::Shape::Rectangle( 20, MODEL_HEIGHT − 60, ((MODEL_WIDTH − 20)/100)* (boss→getHP() * 100/boss→ge
tMaxHP()), MODEL_HEIGHT − 40, sf::Color::White);
  else
    my_rectBossPV = sf::Shape();


  my_rectBossPV.SetColor(sf::Color::Green);
  if(boss→getHP() ≤ 50 * boss→getMaxHP()/100)
    my_rectBossPV.SetColor(sf::Color::Yellow);
  if(boss→getHP() ≤ 30 * boss→getMaxHP()/100)
    my_rectBossPV.SetColor(sf::Color::Red);
}
```



```
#include "Joueur.h"
#include "GameModel.h"
#include "constantes.h"

#include <sstream>

using namespace std;


Joueur::Joueur(int power, TypeTir* typeTir,int vies, int pvMax, int bombs, float shootingSpeed)
:Vaisseau(pvMax, typeTir, shootingSpeed), my_vies(vies), my_power(power),my_bombs(bombs)
{
        my_dx = my_dy = SHEEP_SPEED;
        my_h = SHEEP_HEIGHT * SHEEP_SCALE;
        my_w = SHEEP_WIDTH * SHEEP_SCALE;
        my_x = MODEL_WIDTH/2 − my_w/2;
        my_y = MODEL_HEIGHT − my_h;

        my_numberOfShoots = 1;
}

void Joueur::moveRight()
{if(my_x + my_dx + my_w < MODEL_WIDTH)
        my_x += my_dx;
else
        my_x = MODEL_WIDTH − my_w;}
void Joueur::moveLeft()
{if(my_x − my_dx > 0)
        my_x −= my_dx;
else
        my_x = 0;}
void Joueur::moveDown()
{if(my_y + my_dy + my_h < MODEL_HEIGHT)
        my_y += my_dy;
else
        my_y = MODEL_HEIGHT − my_h;}
void Joueur::moveUp()
{if(my_y − my_dy > 0)
        my_y −= my_dy;
else
        my_y = 0;}



void Joueur::shoot(GameModel* model)
{
        if(my_clock.GetElapsedTime() > my_shootingSpeed)
        {
                switch(my_numberOfShoots)
                {
                        case 1:
                        {       Tir* t = new Tir(my_x, my_y, 0, −my_typeTir→getSpeed(),1, my_typeTir);
                             t→setX(my_x + my_w/2 − t→getW()/2);
                                model→addTir(t);        }
                                break;
                        case 2:
                        {
                                Tir* t1 = new Tir(my_x, my_y, 0, −my_typeTir→getSpeed(),1, my_typeTir);
                                Tir* t2 = new Tir(my_x, my_y, 0, −my_typeTir→getSpeed(),1, my_typeTir);
                                t2→setX(my_x + my_w );
                                model→addTir(t1);
                                model→addTir(t2);
                        }
                                break;
                        case 3:
                        {       Tir* T1 = new Tir(my_x, my_y, 0, −my_typeTir→getSpeed(),1, my_typeTir);
                                Tir* T2 = new Tir(my_x, my_y, 0, −my_typeTir→getSpeed(),1, my_typeTir);
                                Tir* T3 = new Tir(my_x, my_y, 0, −my_typeTir→getSpeed(),1, my_typeTir);
                                T2→setX(my_x + my_w/2 − T2→getW()/2);
                                T3→setX(my_x + my_w − T3→getW());
                                model→addTir(T1);
                                model→addTir(T2);
                                model→addTir(T3);       }
                                break;
                }

                my_clock.Reset();
        }
}

vector<Tir*>::iterator Joueur::isShot(GameModel* model) const
{
        vector<Tir*>::iterator shot = model→getVectorTir()→end();
    vector<Tir*>::iterator it = model→getVectorTir()→begin();
    while( it ≠ model→getVectorTir()→end() ∧ shot ≡ model→getVectorTir()→end())
    {
        if((*it)→getX() + (*it)→getW() > my_x ∧ (*it)→getX() < my_x + my_w ∧ (*it)→getY() + (*it)→getH() > my_y ∧ (*
it)→getY() < my_y+my_h)
        {
            if(¬(*it)→getPlayer())
                shot = it;
        }
        it++;
```

```
    }

   return shot;
}

bool Joueur::isHitByEnemy(GameModel* model)
{
        bool found = false;
        vector<Ennemi*>::iterator it = model→getVectorEnnemis()→begin();
        while(it ≠ model→getVectorEnnemis()→end() ∧ ¬found)
        {
                if((*it)→getX() + (*it)→getW() > my_x ∧ (*it)→getX() < my_x + my_w ∧ (*it)→getY() + (*it)→ getH() >
 my_y ∧ (*it)→getY() < my_y + my_h)
                        if((*it)→getAlive())
                                found = true;

                it++;
        }

        return found;
}

void Joueur::useBomb(GameModel* model)
{
        if(my_bombs > 0)
        {
                for(vector<Ennemi*>::iterator it = model→getVectorEnnemis()→begin(); it≠model→getVectorEnnemis()→end(
); it++)
                {       Boss* b = dynamic_cast<Boss*>(*it);
                        if(b ≡ NULL)
                        {
                                model→setScore(model→getScore() + (*it)→getScore());
                                (*it)→setAlive(false);
                        }
                }
                for(vector<Tir*>::iterator it = model→getVectorTir()→begin(); it ≠ model→getVectorTir()→end(); it++)
                {       model→getDeletedItems()→push_back(*it);
                        delete *it;     }
                model→getVectorTir()→clear();

                my_bombs--;
        }
}

void Joueur::addPower(int power, GameModel* model)
{
        my_power += power;
        if(my_power ≥ 500 ∧ my_numberOfShoots < 3)
                my_numberOfShoots = 3;
        else if(my_power ≥ 300 )
        {
                if( my_typeTir ≠ model→getTirTypes()[2])
                        my_typeTir = model→getTirTypes()[2];
        }
        else if(my_power ≥ 200 ∧ my_numberOfShoots < 2)
                my_numberOfShoots = 2;
        else if(my_power ≥ 100 )
        {
                if(my_typeTir ≠ model→getTirTypes()[1])
                        my_typeTir = model→getTirTypes()[1];
        }
}

void Joueur::addHP(int hp)
{
        if(getHP() + hp ≤ getMaxHP())
                setHP(getHP() + hp);
        else
                setHP(getMaxHP());
}


//Accesseurs
int Joueur::getVies() const
{return my_vies;        }
int Joueur::getPower() const
{   return my_power;    }
int Joueur::getBombs() const
{return my_bombs;       }
int Joueur::getNumberOfShoots() const
{return my_numberOfShoots;      }
void Joueur::setVies(int vies)
{my_vies = vies;        }
void Joueur::setBombs(int bombs)
{my_bombs = bombs;      }
void Joueur::setPower(int i)
{    my_power = i;  }
void Joueur::setNumberOfShoots(int n)
{ my_numberOfShoots = n;        }


string Joueur::getStringBombs() const
```



```
{
        stringstream str;
        str << my_bombs << endl;
        return str.str();
}
string Joueur::getStringPower() const
{
        stringstream str;
        str << my_power << endl;
        return str.str();
}
```

```
#include <iostream>
#include <ctime>
#include "GameView.h"
#include "GameModel.h"

using namespace std;

int main()
{
        srand(time(NULL));

        GameModel *model = NULL;
        GameView * view = new GameView(VIEW_WIDTH, VIEW_HEIGHT,32);

        view->intro();

        bool quit = false;
        int difficulty = 1,initialLives = 3;

        while(quit = false)
        {
                switch(view->menu())
                {
                        case 1:
                                model = new GameModel(7, initialLives, difficulty);
                                view->setModel(model);
                                while(view->treatEvents(quit))
                                {
                                        model->nextStep();
                                        view->synchronize();
                                        view->draw();
                                        view->Display();
                                }
                                delete model;

                                break;
                        case 2:
                                view->viewHighscores(quit);
                                break;
                        case 3:
                                view->options(quit, difficulty, initialLives);
                                break;
                        case 4 :
                                quit = true;
                                break;
                }
        }

  delete view;

        return 0;

}
```



```
#include "MovableElement.h"
#include "constantes.h"

MovableElement::MovableElement()
        :my_x(0),my_y(0),my_dx(3), my_dy(3), my_w(10), my_h(10)
{}
MovableElement::MovableElement(int x, int y, int dx, int dy, int w, int h)
        :my_x(x), my_y(y), my_dx(dx), my_dy(dy), my_w(w), my_h(h)
{}
MovableElement::~MovableElement()
{}

void MovableElement::moveRight()
{my_x += my_dx; }
void MovableElement::moveLeft()
{my_x -= my_dx;}
void MovableElement::moveDown()
{my_y += my_dy;}
void MovableElement::moveUp()
{my_y -= my_dy;}

void MovableElement::move()
{
        my_x += my_dx;
        my_y += my_dy;
}


//Accesseurs
int MovableElement::getX() const
{return my_x;   }
int MovableElement::getY() const
{return my_y;   }
int MovableElement::getDY() const
{return my_dy;  }
int MovableElement::getW() const
{return my_w;   }
int MovableElement::getH() const
{return my_h;   }
void MovableElement::setX(int x)
{my_x = x;      }
void MovableElement::setY(int y)
{my_y = y;      }
```

```
#include "Niveau.h"
#include "GameModel.h"
#include <iostream>

using namespace std;

Niveau::Niveau()
:idNiveau(0)
{}

void Niveau::initNiveau(GameModel* model)
{
        ennemis.clear();
        float t;
        for(int i = 0; i < 20 * (idNiveau+1) * model->getDifficulty() ; i++) // nombre d'ennemis Ã  placer dans la map.
        {
                t = rand() % (15*idNiveau + 30); // temps que dure un niveau : niveau 1 = 30, niveau 2 = 45, niveau 3 =
60 etc...

                int e = rand() % 3 ; //int qui dÃ©termine le type de l'enemi (entre 0 et 2)

                ennemis.insert(pair<float, TypeEnnemi*>(t, model->getEnnemyTypes()[e]));
        }

        ennemis.insert(pair<float, TypeEnnemi*>(15*idNiveau + 30, model->getBossType())); //Ajout du boss en fin de nive
au
        idNiveau++;
}

const multimap<float, TypeEnnemi*>* Niveau::getMap() const
{return & ennemis;      }

int Niveau::getID() const
{return idNiveau;       }
```



```
#include "Tir.h"
#include "constantes.h"

Tir::Tir(int x, int y, int dx, int dy, bool fromPlayer, TypeTir* type)
        :MovableElement(x, y, dx, dy, type->getW(), type->getH()), my_fromPlayer(fromPlayer)
{my_type = type;        }

Tir::~Tir()
{}

bool Tir::getPlayer() const
{return my_fromPlayer;  }
TypeTir* Tir::getType() const
{return my_type;        }
```

```
#include "TypeBonus.h"

TypeBonus::TypeBonus(BonusEffect effect, int value, float frequency, float scale)
:my_effect(effect), my_value(value), my_frequency(frequency), my_scale(scale)
{}

BonusEffect TypeBonus::getEffect() const
{return my_effect;      }
int TypeBonus::getValue() const
{return my_value;       }
float TypeBonus::getFrequency() const
{return my_frequency;   }
float TypeBonus::getScale() const
{return my_scale;       }
```



```
#include "TypeBoss.h"

TypeBoss::TypeBoss(int score, int speed, float shootingSpeed, int hp, int w, int h, TypeTir* typeTir, ShootPattern shoot
Pattern, std::string filename,
                                        float changePatternDelay)
                                        :TypeEnnemi(score, speed, shootingSpeed, hp, w, h, typeTir, shootPattern, filena
me)
{ my_changePatternDelay = changePatternDelay;   }

float TypeBoss::getChangePatternDelay() const
{return my_changePatternDelay;  }
```

```cpp
#include "TypeEnnemi.h"

using namespace std;

TypeEnnemi::TypeEnnemi()
{       my_score =  100;        }

TypeEnnemi::TypeEnnemi(int score, int speed, float shootingSpeed, int hp, int w, int h, TypeTir* typeTir, ShootPattern s
hootPattern, string filename)
{
        my_sprite_filename = filename;
        my_score = score;
        my_speed = speed;
        my_shootingSpeed = shootingSpeed;
        my_hp = hp;
        my_h = h;
        my_w = w;
        my_typeTir = typeTir;
        my_shootPattern = shootPattern;
}


TypeEnnemi::~TypeEnnemi()
{}

//Accesseurs
int TypeEnnemi::getScore() const
{return my_score;       }
int TypeEnnemi::getSpeed() const
{return my_speed;       }
float TypeEnnemi::getShootingSpeed() const
{return my_shootingSpeed;       }
int TypeEnnemi::getHP() const
{return my_hp;  }
int TypeEnnemi::getW() const
{return my_w;   }
int TypeEnnemi::getH() const
{return my_h;   }
string TypeEnnemi::getFilename() const
{return my_sprite_filename;     }
TypeTir* TypeEnnemi::getTypeTir() const
{return my_typeTir;     }
ShootPattern TypeEnnemi::getShootPattern() const
{return my_shootPattern;        }
```



```cpp
#include "TypeTir.h"

using  namespace std;

TypeTir::TypeTir(int w, int h, int damages, int speed, string filename)
        :my_w(w),my_h(h),my_damages(damages), my_speed(speed), my_sprite_filename(filename)
{}



string TypeTir::getFilename() const
{return my_sprite_filename;     }
int TypeTir::getH() const
{return my_h;   }
int TypeTir::getW() const
{return my_w;   }
int TypeTir::getDamages() const
{return my_damages;     }
int TypeTir::getSpeed() const
{return my_speed;       }
```

```cpp
#include <iostream>
#include <vector>
#include "GameModel.h"
#include "Vaisseau.h"
#include "TypeTir.h"
#include "Tir.h"

using namespace std;

// Accesseurs

int Vaisseau::getHP() const
{       return my_hp;   }
int Vaisseau::getMaxHP() const
{       return my_pvMax;        }

TypeTir* Vaisseau::getTypeTir() const
{       return my_typeTir;  }

bool Vaisseau::getAlive() const
{       return my_isAlive;   }

float Vaisseau::getShootingSpeed() const
{       return my_shootingSpeed;        }

sf::Clock* Vaisseau::getClock()
{       return &my_clock;       }

void Vaisseau::setHP(int i)
{    my_hp = i; }

void Vaisseau::setTypeTir(TypeTir* type)
{   my_typeTir = type;  }

void Vaisseau::setAlive(bool b)
{   my_isAlive = b; }

// MÃ©thodes

Vaisseau::Vaisseau(int hp,TypeTir* type, float shootingSpeed)
{
    my_hp = my_pvMax = hp;
    my_typeTir = type;
    my_isAlive = true;
    my_shootingSpeed = shootingSpeed;
}

Vaisseau::~Vaisseau()
{}
```



```cpp
#ifndef BONUS_H
#define BONUS_H

/**
 * \file Bonus.h
 * \brief DÃ©claration de la class Bonus
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ¨T
 */

#include "MovableElement.h"
#include "TypeBonus.h"

/**
 * \class Bonus Bonus.h
 * \brief Classe reprÃ©sentant les bonii Ã  ramasser
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ¨T
 */

class Bonus : public MovableElement
{
        private:
                TypeBonus* my_type; /**< CatÃ©gorie du bonus */

        public:
                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param x : Abscisse du bonus
                 * \param y : OrdonnÃ©e du bonus
                 * \param type : Type du bonus
                 */
                Bonus(int x, int y, TypeBonus* type);

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_type
                 *
                 * \return : Le type du Bonus
                 */
                TypeBonus* getType() const;

};

#endif
```

```
#ifndef BOSS_H
#define BOSS_H

/**
 * \file Boss.h
 * \brief DÃ©claration de la class Boss
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

#include "Ennemi.h"
#include "TypeBoss.h"

/**
 * \class Boss Boss.h
 * \brief Classe reprÃ©sentant les boss de fin de niveau
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

class Boss : public Ennemi
{
        private:
                TypeBoss* my_type; /**< Type du bonus */

                sf::Clock my_changePatternClock; /**< Clock (sert Ã  changer le pattern rÃ©guliÃ¨rement) */
                float my_changePatternDelay; /**< DÃ©lai entre deux changements de pattern */

                ShootPattern my_actualShootPattern; /**< Pattern actuel */


        public:
                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param type : Type du boss
                 */
                Boss(TypeBoss* type);

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_actualShootPattern
                 *
                 * \return : Le pattern actuellement utilisÃ©
                 */
                ShootPattern getShootPattern() const;

                /**
                 * \brief Changer le pattern
                 *
                 * MÃ©thode permettant de changer le pattern actuellement utilisÃ©
                 */
                void changePattern();

                /**
                 * \brief DÃ©placer le boss
                 *
                 * MÃ©thode permettant de dÃ©placer le Boss
                 */
                void move();

};

#endif
```



```
/**
 * \file constantes.h
 * \brief DÃ©claration des constantes utilisÃ©es
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

#define MODEL_WIDTH 768 /**< Largeur du ModÃ¨le */
#define MODEL_HEIGHT 768 /**< Hauteur du ModÃ¨le */

#define VIEW_WIDTH 768 /**< Largeur de la Vue */
#define VIEW_HEIGHT 768 /**< Hauteut de la Vue */

#define SHEEP_WIDTH 200 /**< Largeur du Joueur */
#define SHEEP_HEIGHT 309 /**< Hauteur du Joueur */
#define SHEEP_SCALE 0.2 /**< Echelle du Joueur */

#define ENEMY_SCALE 0.3 /**< Echelle des Ennemi */

#define SHEEP_SPEED 10 /**< Vitesse du Joueur */
#define MOVABLE_ELEMENT_SPEED 10 /**< Vitesse par dÃ©faut des MovableElement */

#define NB_ENNEMY_TYPES 3 /**< Nombre de TypeEnnemi */
#define NB_TIRS_TYPES 7 /**< Nombre de TypeTir */
#define NB_BONUS_TYPES 5 /**< Nombre de TypeBonus */

//Pour se repÃ©rer dans le tableau de strings
#define NB_STRINGS 26 /**< Nombre d'entrÃ©es du tableau contenant les textes du jeu */

#define VIES 0 /**< Index de la chaÃ®ne "Vies" */
#define PV 1 /**< Index de la chaÃ®ne "PV" */
#define NIVEAU 2 /**< Index de la chaÃ®ne "Niveau" */
#define BOMBES 3 /**< Index de la chaÃ®ne "Bombes" */
#define PUISSANCE 4 /**< Index de la chaÃ®ne "Puissance" */
#define JOUER 5 /**< Index de la chaÃ®ne "Jouer" */
#define HIGHSCORES 6 /**< Index de la chaÃ®ne "Meilleurs scores" */
#define SETTINGS 7 /**< Index de la chaÃ®ne "ParamÃ¨tres" */
#define QUIT 8 /**< Index de la chaÃ®ne "Quitter" */
#define PAUSE 9 /**< Index de la chaÃ®ne "Pause" */
#define RESUME 10 /**< Index de la chaÃ®ne "Continuer" */
#define BACKTOMENU 11 /**< Index de la chaÃ®ne "Retourner au Menu" */
#define NOSCORE 12 /**< Index de la chaÃ®ne "Pas de score" */
#define ENTERNAME 13 /**< Index de la chaÃ®ne "Entrez votre nom" */
#define FINISHLEVEL 14 /**< Index de la chaÃ®ne "Vous avez fini un niveau !" */
#define GAMEOVER 15 /**< Index de la chaÃ®ne "Partie TerminÃ©e !" */
#define DEAD 16 /**< Index de la chaÃ®ne "Vous avez perdu une vie ! " */
#define SOUNDLEVEL 17 /**< Index de la chaÃ®ne "Volume sonore" */
#define DIFFICULTY 18 /**< Index de la chaÃ®ne "DifficultÃ©" */
#define LIVESNUMBER 19 /**< Index de la chaÃ®ne "Nombre de vies" */
#define LANGUAGE 20 /**< Index de la chaÃ®ne "Langue" */
#define NORMAL 21 /**< Index de la chaÃ®ne "Normal" */
#define HARD 22 /**< Index de la chaÃ®ne "Difficile" */
#define VERYHARD 23 /**< Index de la chaÃ®ne "TrÃ¨s difficile" */
#define MYLANGUE 24 /**< Index de la chaÃ®ne indiquant la langue */
#define SOUNDPACK 25 /**< Index de la chaÃ®ne "Pack de son" */
```

```
#ifndef ENNEMI_H
#define ENNEMI_H

/**
 * \file Ennemi.h
 * \brief DÃ©claration de la class Ennemi
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

#include <SFML/Window.hpp>
#include "Vaisseau.h"
#include "TypeEnnemi.h"
#include "Horloge.h"

/**
 * \class Ennemi Ennemi.h
 * \brief Classe reprÃ©sentant les ennemis du joueur
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

class Ennemi : public Vaisseau
{
	private :
		TypeEnnemi* my_type; /**< Type d'ennemi */

	public :

		/**
		 * \brief Constructeur par dÃ©faut
		 */
		Ennemi();

		/**
		 * \brief Constructeur paramÃ©trÃ©
		 *
		 * \param type : Type d'ennemi
		 */
		Ennemi(TypeEnnemi* type);

		/**
		 * \brief Destructeur
		 */
		virtual ~Ennemi();


		/**
		 * \brief Accesseur
		 *
		 * Accesseur de my_type.my_score
		 *
		 * \return : Les points rapportÃ©s par l'ennemi
		 */
		int getScore() const;

		/**
		 * \brief Accesseur
		 *
		 * Accesseur de my_type
		 *
		 * \return : Le type d'ennemi
		 */
		TypeEnnemi* getType() const;

		/**
		 * \brief Accesseur
		 *
		 * Accesseur de my_type.my_shootPattern
		 *
		 * \return : Le pattern utilisÃ© par l'ennemi
		 */
		virtual ShootPattern getShootPattern() const;


		/**
		 * \brief Faire tirer l'ennemi
		 *
		 * \param model : ModÃ©le du jeu
		 */
		void shoot(GameModel* model);

		/**
		 * \brief Indiquer le tir touchant l'ennemi
		 *
		 * \param model : ModÃ©le du jeu
		 * \return : Le tir touchant l'ennemi
		 */
		std::vector<Tir*>::iterator isShot(GameModel* model) const;


		/**
		 * \brief DÃ©placer l'ennemi
```



```
		 */
		void move();
};

#endif
```

```
#ifndef GAMEMODEL_H
#define GAMEMODEL_H

/**
 * \file GameModel.h
 * \brief DÃ©claration de la classe GameModel
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

#include "Tir.h"
#include "Ennemi.h"
#include "Boss.h"
#include "Joueur.h"
#include "Bonus.h"
#include "Niveau.h"
#include "Horloge.h"

#include <vector>
#include <string>
#include <sstream>

using namespace std;

/**
 * \class GameModel GameModel.h
 * \brief Classe reprÃ©sentant le ModÃ¨le du jeu
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

class GameModel
{
        private :

                int my_score;
                int my_viesMax;
                int my_difficulty;

                Joueur *my_ship;
                vector<Tir*> my_tirs;
                vector<Ennemi*> my_ennemis;
                vector<Bonus*> my_bonuses;
                vector<MovableElement*> my_deletedItems;

                TypeEnnemi* my_ennemi_types[NB_ENNEMY_TYPES];
                TypeTir* my_tirs_types[NB_TIRS_TYPES];
                TypeBonus* my_bonuses_types[NB_BONUS_TYPES];
                TypeBoss* my_boss_type;

                Boss *b;


                Niveau niveau;
                map<float, TypeEnnemi*>::const_iterator itNiveau;
                bool my_nextLevel;

                Horloge my_clock;


        public :

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param viesMax : Nombre maximum de vies du Joueur
                 * \param initialLives : Nombre initial de vies du Joueur
                 * \param difficulty : DifficultÃ© du jeu
                 */
                GameModel(int viesMax, int initialLives, int difficulty);

                /**
                 * \brief Destructeur
                 */
                ~GameModel();

                /**
                 * \brief Passer Ã  "l'Ã©tape" suivante
                 *
                 * MÃ©thode permettant de faire progresser le jeu
                 */
                void nextStep();


                /**
                 * \brief Ajouter un Ennemi
                 *
                 * \param type : Type de l'Ennemi Ã  ajouter
                 */
                void addEnnemy(TypeEnnemi* type);

                /**
                 * \brief Ajouter un Tir
```



```
                 *
                 * \param t : Tir Ã  ajouter
                 */
                void addTir(Tir* t);

                /**
                 * \brief Ajouter un Bonus
                 *
                 * \param x : Abscisse du Bonus Ã  ajouter
                 * \param y : OrdonnÃ©e du Bonus Ã  ajouter
                 */
                void addBonus(int x, int y);


                /**
                 * \brief Supprimer un Ennemi
                 *
                 * \param ennemi : Adresse de l'Ennemi Ã  supprimer
                 */
                void removeEnnemy(Ennemi* ennemi);

                /**
                 * \brief Supprimer un Ennemi
                 *
                 * \param it : ItÃ©rateur vers l'Ennemi Ã  supprimer
                 */
                void removeEnnemy(std::vector<Ennemi*>::iterator it);


                /**
                 * \brief Sauvegarder le score
                 *
                 * \param name : Pseudo Ã  afficher
                 */
                void saveScore(string name);


                /**
                 * \brief Passer au Niveau suivant
                 *
                 * Vide le vector des Tir, rÃ©initialise la Clock et redÃ©finit le Boss
                 */
                void changeNiveau();

                /**
                 * \brief Tuer le Joueur
                 *
                 * MÃ©thode permettant de gÃ©rer la mort du Joueur
                 */
                void gameOver();


                //Accesseurs

                /**
                 * \brief Accesseur
                 *
                 * MÃ©thode rÃ©cupÃ©rant la position du Joueur
                 *
                 * \param x : (Sortie seulement) Abscisse du Joueur
                 * \param y : (Sortie seulement) OrdonnÃ©e du Joueur
                 */
                void getShipPos(int &x, int &y) const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_nextLevel
                 *
                 * \return : Si l'on passe ou non au niveau suivant (\b true si oui, \b sinon)
                 */
                bool getNextLevel() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de niveau.my_id
                 *
                 * \return : L'identifiant du Niveau en cours
                 */
                int getLevelID() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_score
                 *
                 * \return : Le score actuel
                 */
                int getScore() const;

                /**
                 * \brief Accesseur
```

```
     *
     * Accesseur de my_difficulty
     *
     * \return : La difficultÃ© du jeu
     */
    int getDifficulty() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de niveau.my_id
     *
     * \return : L'identifiant du Niveau en cours, sous forme de chaÃ®ne de caractÃ¨res
     */
    string getStringLevelID() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_score
     *
     * \return : Le score actuel, sous forme de chaÃ®ne de caractÃ¨res
     */
    string getStringScore() const;


    /**
     * \brief Accesseur
     *
     * Accesseur de my_ennemi_types
     *
     * \return : Le tableau contenant les diffÃ©rents TypeEnnemi
     */
    TypeEnnemi* const* getEnnemyTypes() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_tirs_types
     *
     * \return : Le tableau contenant les diffÃ©rents TypeTir
     */
    TypeTir* const*  getTirTypes() const ;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_boss_types
     *
     * \return : Le tableau contenant les diffÃ©rents TypeBoss
     */
    TypeBoss* getBossType() const;


    /**
     * \brief Accesseur
     *
     * Accesseur de my_ship
     *
     * \return : Le Joueur
     */
    Joueur*  getPlayerShip() const;


    /**
     * \brief Accesseur
     *
     * Accesseur de my_ennemis
     *
     * \return : Le vecteur contenant les Ennemi prÃ©sents
     */
    vector<Ennemi*>* getVectorEnnemis();

    /**
     * \brief Accesseur
     *
     * Accesseur de my_tirs
     *
     * \return : Le vecteur contenant les Tir prÃ©sents
     */
    vector<Tir*>* getVectorTir();

    /**
     * \brief Accesseur
     *
     * Accesseur de my_bonuses
     *
     * \return : Le vecteur contenant les Bonus prÃ©sents
     */
    const vector<Bonus*> & getVectorBonuses();

    /**
     * \brief Accesseur
```



```
     *
     * Accesseur de my_tirs
     *
     * \return : Le vecteur contenant les objets Ã  supprimer
     */
    vector<MovableElement*>* getDeletedItems();


    /**
     * \brief Accesseur
     *
     * Accesseur de my_clock
     *
     * \return : L'Horloge du jeu
     */
    Horloge*  getClock();


    /**
     * \brief DÃ©finir le score
     * \param score : Nouveau score
     */
    void setScore(int score);

    /**
     * \brief DÃ©finir si l'on passe au niveau suivant
     * \param nextLvl : \b true si l'on passe au niveau suivant, \b false sinon
     */
    void setNextLevel(bool nextLvl);

};

#endif
```

```
#ifndef GAME_VIEW_H
#define GAME_VIEW_H

/**
 * \file GameView.h
 * \brief DÃ©claration de la classe GameView
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ¯T
 */

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <map>
#include <vector>
#include "GraphicElement.h"
#include "MovableElement.h"
#include "TypeEnnemi.h"
#include "constantes.h"
#include "Interface.h"

class GameModel;

/**
 * \class GameView GameView.h
 * \brief Classe reprÃ©sentant la Vue du jeu
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ¯T
 */

class GameView {
        private :
                int my_w, my_h;

                sf::RenderWindow *my_window;
                sf::Image *my_background_image;
        sf::Image *my_background_parallax;
                sf::Image *my_ship_image;
                sf::Image *my_shoot_image;
                sf::Image *my_boom_image;
                sf::Image *my_revive_image;
                sf::Image *my_bonus_image;
                std::map<TypeEnnemi* const, sf::Image*> my_ennemies_images; //Map qui contient les images des ennemis en
 fonction de leur type
                std::map<TypeTir* const, sf::Image*> my_tirs_images;

                GraphicElement my_background_sprite, my_background_sprite2;
        GraphicElement my_background_parallax_sprite, my_background_parallax_sprite2;

                Interface my_hud;
                std::string my_strings[NB_STRINGS];
                int my_langue;

                 //map qui contient tout les sprites;
                std::map<MovableElement * const, GraphicElement *> my_elementToGraphicElement;

                //sons
                sf::SoundBuffer *my_titlethemeBuffer;
                sf::SoundBuffer *my_deathmonsterBuffer;
                sf::SoundBuffer *my_deathbossBuffer;
                sf::SoundBuffer *my_nextLevelBuffer;
                sf::SoundBuffer *my_gameoverBuffer;
                sf::SoundBuffer *my_shotBuffer;

                sf::Music my_maintheme;
                sf::Sound my_titletheme;
                sf::Sound my_deathmonster;
                sf::Sound my_deathboss;
                sf::Sound my_nextLevel;
                sf::Sound my_gameover;
                sf::Sound my_shotsound;

                int my_soundLevel;
                int my_soundpack;

                GameModel *my_model;

                sf::Clock my_clock; // Clock qui sert pour le tir

        public :

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param w : Largeur de la Vue
                 * \param h : Hauteur de la Vue
                 * \param bpp : Bits par pixel (profondeur des couleurs)
                 */
                GameView(int w, int h, int bpp);

                /**
                 * \brief Destructeur
                 */
                ~GameView();
```



```

                /**
                 * \brief DÃ©finir le ModÃ¨le
                 * \param model : GameModel Ã  utiliser
                 */
                void setModel(GameModel * model);


                /**
                 * \brief Mettre Ã  jour la langue
                 */
                void updateLanguage();

                /**
                 * \brief Mettre Ã  jour le volume sonore
                 */
                void updateVolume();

                /**
                 * \brief Mettre Ã  jour les sons utilisÃ©s
                 */
                void updateSound();


                /**
                 * \brief Faire dÃ©filer l'arriÃ¨re-plan
                 */
                void bgscroll();

                /**
                 * \brief Dessiner une explosion
                 * \param elem : Sprite de l'Ennemi explosant
                 * \param adr : Ennemi explosant
                 */
                void explosion(GraphicElement &elem, Ennemi* adr);


                /**
                 * \brief Dessiner tous les Ã©lÃ©ments dans la Vue
                 */
                void draw();

                /**
                 * \brief Afficher les Ã©lÃ©ments dans une fenÃªtre
                 */
                void Display();

                /**
                 * \brief Dessiner une transition
                 *
                 * MÃ©thode permettant d'afficher un texte et un compte Ã  rebours aprÃ¨s la mort du Joueur ou avant le
dÃ©but d'un Niveau par exemple
                 *
                 * \param text : Texte de la transition
                 * \param t : DurÃ©e de la transition (en secondes)
                 */
                void transition(std::string text, float t);

                /**
                 * \brief Synchroniser la Vue avec le ModÃ¨le
                 * \bug Une technique plus optimisÃ©e a Ã©tÃ© trouvÃ©e pour cette mÃ©thode, mais elle entraÃ®ne parfois
des erreurs de segmentation, pour une raison inconnue
                 */
                void synchronize();

                /**
                 * \brief Traite les Ã©vÃ©nements de la SFML
                 * \param quit : (Sortie seulement) Ã©tat du programme
                 */
                bool treatEvents(bool &quit);


                /**
                 * \brief Dessiner l'Ã©cran de pause
                 */
                int pause();


                /**
                 * \brief Dessiner l'introduction du jeu
                 */
                void intro();

                /**
                 * \brief Dessiner le menu principal
                 */
                int menu();

                /**
                 * \brief Dessiner le menu des options
                 * \param quit : (Sortie seulement) Ã©tat du programme
                 * \param difficulty : (Sortie seulement) DifficultÃ© du jeu
```

```
                 * \param initialLives : (Sortie seulement) Nombre initial de vies du Joueur
                 */
                void options(bool &quit, int &difficulty, int &initialLives);

                /**
                 * \brief Dessiner le menu des meilleurs scores
                 * \param quit : (Sortie seulement) Ã©tat du programme
                 * \bug Pour une raison inconnue, un Event::Closed se produit Ã  l'ouverture du menu. Pour pallier Ã  Ã§
a, il a fallu empÃªcher de fermer le programme "normalement" une fois sur cet Ã©cran (l'utilisation de la touche Escape
 est toujours possible cependant)
                 */
                void viewHighscores(bool &quit);

                /**
                 * \brief Dessiner la demande de son pseudo Ã  l'utilisateur
                 */
                void askName();


};

#endif
```



```
#ifndef GRAPHICELEMENT_H
#define GRAPHICELEMENT_H

/**
 * \file GraphicElement.h
 * \brief DÃ©claration de la classe GraphicElement
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ«T
 */

#include <SFML/Graphics.hpp>

/**
 * \class GraphicElement GraphicElement.h
 * \brief Classe reprÃ©sentant les Ã©lÃ©ments Ã  afficher
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ«T
 */

class GraphicElement : public sf::Sprite
{
        private:
                int _w, _h;
                bool _visible;

        public:

                /**
                 * \brief Constructeur par dÃ©faut
                 */
                GraphicElement();

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param image : Image Ã  charger
                 * \param x : Abscisse de l'emplacement de l'Ã©lÃ©ment Ã  afficher dans la Vue
                 * \param y : OrdonnÃ©e de l'emplacement de l'Ã©lÃ©ment Ã  afficher dans la Vue
                 * \param w : Largeur de l'image Ã  charger
                 * \param h : Hauteur de l'image Ã  charger
                 */
                GraphicElement(sf::Image * image, int x, int y, int w, int h);

                /**
                 * \brief Destructeur
                 */
                virtual ~GraphicElement();


                /**
                 * \brief Dessiner l'Ã©lÃ©ment Ã  afficher
                 *
                 * MÃ©thode permettant de dessiner l'Ã©lÃ©ment Ã  afficher dans une fenÃªtre
                 *
                 * \param _window : FenÃªtre oÃ¹ dessiner l'Ã©lÃ©ment Ã  afficher
                 */
                virtual void draw(sf::RenderWindow * _window);


                /**
                 * \brief Modifier la position de l'Ã©lÃ©ment Ã  afficher
                 * \param pos : Vecteur contenant les nouvelles coordonnÃ©es de l'Ã©lÃ©ment Ã  afficher
                 */
                void setPosition(const sf::Vector2f & pos);

                /**
                 * \brief Modifier la position de l'Ã©lÃ©ment Ã  afficher
                 * \param x : Nouvelle abscisse de l'Ã©lÃ©ment Ã  afficher
                 * \param y : Nouvelle ordonnÃ©e de l'Ã©lÃ©ment Ã  afficher
                 */
                void setPosition(int x, int y);

                /**
                 * \brief Modifier les dimensions de l'Ã©lÃ©ment Ã  afficher
                 * \param w : Nouvelle largeur de l'Ã©lÃ©ment Ã  afficher
                 * \param h : Nouvelle hauteur de l'Ã©lÃ©ment Ã  afficher
                 */
                void resize(int w, int h);


                /**
                 * \brief Accesseur
                 *
                 * Accesseur de _visible
                 *
                 * \return : La visibilitÃ© actuelle de l'Ã©lÃ©ment
                 */
                bool getVisible() const;

                /**
                 * \brief DÃ©finir la visibilitÃ© de l'Ã©lÃ©ment Ã  afficher
                 * \param visible : Nouvelle visibilitÃ© de l'Ã©lÃ©ment Ã  afficher
                 */
```

```
                void setVisible(bool visible);
};
#endif
```



```
#ifndef HORLOGE_H
#define HORLOGE_H

/**
 * \file Horloge.h
 * \brief DÃ©claration de la classe Horloge
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

#include <ctime>

/**
 * \class Horloge Horloge.h
 * \brief Classe reprÃ©sentant le temps dans le jeu
 *
 * Remplace la classe Clock de la SFML 1.6, afin notamment d'implÃ©menter une pause
 *
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

class Horloge
{
        private:
                time_t m_startTime;
                timespec tp;
                float m_buffer;
                bool m_paused;
        public:

                /**
                 * \brief Constructeur par dÃ©faut
                 */
                Horloge();

                /**
                 * \brief Retourner le temps Ã©coulÃ©
                 * \return : Le temps Ã©coulÃ© depuis le dÃ©marrage de l'horloge
                 */
                float getTime();

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de m_paused
                 *
                 * \return : L'Ã©tat de l'horloge (\b true si l'horloge est en pause, \b false sinon)
                 */
                bool getStatus() const;


                /**
                 * \brief DÃ©marrer l'horloge
                 */
                void start();

                /**
                 * \brief Mettre l'horloge en pause
                 */
                void pause();

                /**
                 * \brief ArrÃªter l'horloge
                 *
                 * Met l'horloge en pause et la rÃ©initialise
                 *
                 */
                void stop();

                /**
                 * \brief RÃ©initialiser l'horloge
                 */
                void reset();


};

#endif
```

```
#ifndef INTERFACE_H
#define INTERFACE_H

/**
 * \file Interface.h
 * \brief DÃ©claration de la classe Interface
 * \author Quentin HARSCOÃM-^KT
 */

#include <SFML/Graphics.hpp>
#include <string>

#include "GameModel.h"
#include "GraphicElement.h"
#include "constantes.h"

/**
 * \class Interface Interface.h
 * \brief Classe reprÃ©sentant le HUD (affichage tÃªte haute)
 * \author Quentin HARSCOÃM-^KT
 */

class Interface
{
	private:
		//HUD
		sf::String my_HUDsfStrings[6];

		sf::Image* my_revive_image;
		GraphicElement my_revive_sprite_template;
		int my_nbVies;

		sf::Shape my_rectPV;
		sf::Shape my_rectPVFond;

		sf::Shape my_rectBossPV;

		sf::Font my_font;
	public:

		/**
		 * \brief Constructeur par dÃ©faut
		 */
		Interface();

		/**
		 * \brief Destructeur
		 */
		~Interface();


		/**
		 * \brief Dessiner le HUD
		 *
		 * MÃ©thode permettant de dessiner le HUD dans une fenÃªtre
		 *
		 * \param _window : FenÃªtre oÃ¹ dessiner le HUD
		 */
		void drawHUD(sf::RenderWindow * _window);


		/**
		 * \brief Mettre Ã  jour le HUD
		 *
		 * \param model : ModÃ¨le du jeu
		 * \param strings[] : Tableau contenant les textes du HUD (utilisÃ© pour la traduction)
		 */
		void update(GameModel* model, string strings[]);

		/**
		 * \brief Mettre Ã  jour la barre de vie du boss
		 *
		 * \param boss : Boss affrontÃ©
		 */
		void updateBossPV(Boss* boss);


};

#endif
```



```
#ifndef JOUEUR_H
#define JOUEUR_H

/**
 * \file Joueur.h
 * \brief DÃ©claration de la classe Joueur
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

#include "Vaisseau.h"
#include "Ennemi.h"
#include <string>

class Tir;
class GameModel;

/**
 * \class Joueur Joueur.h
 * \brief Classe reprÃ©sentant le vaisseau contrÃ´lÃ©
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

class Joueur : public Vaisseau
{
	private :
		int my_vies;
		int my_power;
		int my_bombs;
		int my_numberOfShoots;

	public :

		/**
		 * \brief Constructeur paramÃ©trÃ©
		 *
		 * \param power : Puissance initiale du vaisseau contrÃ´lÃ©
		 * \param typeTir : Type de tir initial du vaisseau contrÃ´lÃ©
		 * \param vies : Nombre initial de vies du vaisseau contrÃ´lÃ©
		 * \param pvMax : Nombre maximum de points de vie du vaisseau contrÃ´lÃ©
		 * \param bombs : Nombre initial de bombes du vaisseau contrÃ´lÃ©
		 * \param shootingSpeed : Vitesse de tir initiale du vaisseau contrÃ´lÃ©
		 */
		Joueur(int power, TypeTir* typeTir, int vies, int pvMax, int bombs, float shootingSpeed);


		/**
		 * \brief DÃ©placer le vaisseau contrÃ´lÃ© vers la gauche
		 */
		void moveLeft();

		/**
		 * \brief DÃ©placer le vaisseau contrÃ´lÃ© vers la droite
		 */
		void moveRight();

		/**
		 * \brief DÃ©placer le vaisseau contrÃ´lÃ© vers le bas
		 */
		void moveDown();

		/**
		 * \brief DÃ©placer le vaisseau contrÃ´lÃ© vers le haut
		 */
		void moveUp();


		/**
		 * \brief Faire tirer le vaisseau contrÃ´lÃ©
		 *
		 * \param model : ModÃ©le du jeu
		 */
		void shoot(GameModel* model);

		/**
		 * \brief Indiquer le tir touchant le vaisseau
		 *
		 * \param model : ModÃ©le du jeu
		 * \return : Le tir touchant le vaisseau contrÃ´lÃ©
		 */
		std::vector<Tir*>::iterator isShot(GameModel* model) const;

		/**
		 * \brief Indiquer si le vaisseau contrÃ´lÃ© est touchÃ©
		 *
		 * \param model : ModÃ©le du jeu
		 * \return : L'Ã©tat du vaisseau contrÃ´lÃ© (\b true si le vaisseau contrÃ´lÃ© est touchÃ©, \b false sin
on)
		 */
		bool isHitByEnemy(GameModel* model);

		/**
```

```
     * \brief Utiliser une bombe
     *
     * MÃ©thode permettant de dÃ©truire tous les ennemis Ã  l'Ã©cran
     *
     * \param model : ModÃ¨le du jeu
     */
    void useBomb(GameModel* model);


    /**
     * \brief Ajouter de la puissance
     *
     * \param power : Puissance Ã  ajouter
     * \param model : ModÃ¨le du jeu
     */
    void addPower(int power, GameModel* model);

    /**
     * \brief Ajouter des points de vie
     *
     * \param hp : Points de vie Ã  ajouter
     */
    void addHP(int hp);


    /**
     * \brief Accesseur
     *
     * Accesseur de my_vies
     *
     * \return : Le nombre de vies actuel du vaisseau contrÃ´lÃ©
     */
    int getVies() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_power
     *
     * \return : La puissance actuelle du vaisseau contrÃ´lÃ©
     */
    int getPower() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_bombs
     *
     * \return : La quantitÃ© de bombes actuelle du vaisseau contrÃ´lÃ©
     */
    int getBombs() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_numberOfShoots
     *
     * \return : Le nombre de tirs simultanÃ©s du vaisseau contrÃ´lÃ© (dÃ©pend de la puissance)
     */
    int getNumberOfShoots() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_bombs
     *
     * \return : La quantitÃ© de bombes actuelle du vaisseau contrÃ´lÃ©, en chaÃ®ne de caractÃ¨res
     */
    std::string getStringBombs() const;

    /**
     * \brief Accesseur
     *
     * Accesseur de my_power
     *
     * \return : La puissance actuelle du vaisseau contrÃ´lÃ©, en chaÃ®ne de caractÃ¨res
     */
    std::string getStringPower() const;


    /**
     * \brief DÃ©finir le nombre de vies du vaisseau contrÃ´lÃ©
     */
    void setVies(int vies);

    /**
     * \brief DÃ©finir la quantitÃ© de bombes du vaisseau contrÃ´lÃ©
     */
    void setBombs(int bombs);

    /**
     * \brief DÃ©finir la puissance du vaisseau contrÃ´lÃ©
     */
```



```
    void setPower(int i);

    /**
     * \brief DÃ©finir le nombre de tirs simultanÃ©s du vaisseau contrÃ´lÃ©
     */
    void setNumberOfShoots(int n);

};

#endif
```

```
#ifndef MOVABLE_ELEMENT_H
#define MOVABLE_ELEMENT_H

/**
 * \file MovableElement.h
 * \brief DÃ©claration de la classe MovableElement
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

/**
 * \class MovableElement MovableElement.h
 * \brief Classe reprÃ©sentant les Ã©lÃ©ments mobiles
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ‹T
 */

class MovableElement
{
        protected :
                int my_x; /**< Abscisse de l'Ã©lÃ©ment mobile */
                int my_y; /**< OrdonnÃ©e de l'Ã©lÃ©ment mobile */
                int my_dx; /**< Vitesse (abscisse) de l'Ã©lÃ©ment mobile */
                int my_dy; /**< Vitesse (ordonnÃ©e) de l'Ã©lÃ©ment mobile */
                int my_w; /**< Largeur de l'Ã©lÃ©ment mobile */
                int my_h; /**<  Hauteur de l'Ã©lÃ©ment mobile*/


        public :

                /**
                 * \brief Constructeur par dÃ©faut
                 */
                MovableElement();

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param x : Abscisse de l'emplacement de l'Ã©lÃ©ment mobile dans le ModÃ©le
                 * \param y : OrdonnÃ©e de l'emplacement de l'Ã©lÃ©ment mobile dans le ModÃ©le
                 * \param dx : Vitesse (abscisse) de l'Ã©lÃ©ment mobile
                 * \param dy : Vitesse (ordonnÃ©e) de l'Ã©lÃ©ment mobile
                 * \param w : Largeur de l'Ã©lÃ©ment mobile
                 * \param h : Hauteur de l'Ã©lÃ©ment mobile
                 */
                MovableElement(int x, int y, int dx, int dy, int w, int h);

                /**
                 * \brief Destructeur
                 */
                virtual ~MovableElement();


                /**
                 * \brief DÃ©placer l'Ã©lÃ©ment mobile vers la gauche
                 */
                virtual void moveLeft();

                /**
                 * \brief DÃ©placer l'Ã©lÃ©ment mobile vers la droite
                 */
                virtual void moveRight();

                /**
                 * \brief DÃ©placer l'Ã©lÃ©ment mobile vers le bas
                 */
                virtual void moveDown();

                /**
                 * \brief DÃ©placer l'Ã©lÃ©ment mobile vers le haut
                 */
                virtual void moveUp();


                /**
                 * \brief DÃ©placer l'Ã©lÃ©ment mobile
                 *
                 * MÃ©thode permettant de dÃ©placer l'Ã©lÃ©ment mobile en abscisse et en ordonnÃ©e
                 */
                virtual void move();

                //Accesseurs

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_x
                 *
                 * \return : L'abscisse de l'Ã©lÃ©ment mobile
                 */
                int getX() const;

                /**
```



```
                 * \brief Accesseur
                 *
                 * Accesseur de my_y
                 *
                 * \return : L'ordonnÃ©e de l'Ã©lÃ©ment mobile
                 */
                int getY() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_dy
                 *
                 * \return : La vitesse (ordonnÃ©e) de l'Ã©lÃ©ment mobile
                 */
                int getDY() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_w
                 *
                 * \return : La largeur de l'Ã©lÃ©ment mobile de l'Ã©lÃ©ment mobile
                 */
                int getW() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_h
                 *
                 * \return : La hauteur de l'Ã©lÃ©ment mobile de l'Ã©lÃ©ment mobile
                 */
                int getH() const ;

                /**
                 * \brief DÃ©finir l'abscisse de l'Ã©lÃ©ment mobile
                 */
                void setX(int x);

                /**
                 * \brief DÃ©finir l'ordonnÃ©e de l'Ã©lÃ©ment mobile
                 */
                void setY(int y);

};

#endif
```

```
#ifndef NIVEAU_H
#define NIVEAU_H

/**
 * \file Niveau.h
 * \brief DÃ©claration de la classe Niveau
 * \author Quentin HARSCOÃM~^KT
 */

#include <map>
#include "TypeEnnemi.h"

class GameModel;

/**
 * \class Niveau Niveau.h
 * \brief Classe reprÃ©sentant les niveaux du jeu
 * \author Quentin HARSCOÃM~^KT
 */

class Niveau
{
	private :

		int idNiveau;
		std::multimap<float, TypeEnnemi*> ennemis; // float : Instant (en secondes) oÃ¹ l'ennemi apparait, TypeEnnemi* : Le type de l'ennemi

	public :

		/**
		 * \brief Constructeur par dÃ©faut
		 */
		Niveau();

		/**
		 * \brief Initialise le niveau
		 * \param model : ModÃ¨le du jeu
		 */
		void initNiveau(GameModel* model);

		/**
		 * \brief Accesseur
		 *
		 * Accesseur de ennemis
		 *
		 * \return : L'adresse de la multimap contenant les instant oÃ¹ les ennemis apparÃ®ssent, ainsi que les types de ces derniers
		 */
		const std::multimap<float, TypeEnnemi*>* getMap() const;

		/**
		 * \brief Accesseur
		 *
		 * Accesseur de idNiveau
		 *
		 * \return : L'identifiant du niveau
		 */
		int getID() const;
};



#endif
```



```
#ifndef TIR_H
#define TIR_H

/**
 * \file Tir.h
 * \brief DÃ©claration de la classe Tir
 * \author Quentin HARSCOÃM~^KT
 */

#include "MovableElement.h"
#include "TypeTir.h"

/**
 * \class Tir Tir.h
 * \brief Classe reprÃ©sentant les tirs des Vaisseau
 * \author Quentin HARSCOÃM~^KT
 */

class Tir : public MovableElement
{
	private :
		bool my_fromPlayer;
		TypeTir* my_type;

	public :

		/**
		 * \brief Constructeur paramÃ©trÃ©
		 *
		 * \param x : Abscisse de l'emplacement du tir dans le ModÃ©le
		 * \param y : OrdonnÃ©e de l'emplacement du tir dans le ModÃ©le
		 * \param fromPlayer : Origine du tir (\b true si le tir vient du joueur, \b false sinon)
		 * \param type : Type du tir
		 */
		Tir(int x, int y, bool fromPlayer, TypeTir* type);

		/**
		 * \brief Constructeur paramÃ©trÃ©
		 *
		 * \param x : Abscisse de l'emplacement du tir dans le ModÃ©le
		 * \param y : OrdonnÃ©e de l'emplacement du tir dans le ModÃ©le
		 * \param dx : Vitesse (abscisse) du tir
		 * \param dy : Vitesse (ordonnÃ©e) du tir
		 * \param fromPlayer : Origine du tir (\b true si le tir vient du joueur, \b false sinon)
		 * \param type : Type du tir
		 */
		Tir(int x, int y, int dx, int dy, bool fromPlayer, TypeTir* type);

		/**
		 * \brief Destructeur
		 */
		virtual ~Tir();


		/**
		 * \brief Accesseur
		 *
		 * Accesseur de my_fromPlayer
		 *
		 * \return : L'origine du tir (\b true si le tir vient du joueur, \b false sinon)
		 */
		bool getPlayer() const;

		/**
		 * \brief Accesseur
		 *
		 * Accesseur de my_type
		 *
		 * \return : Le type du tir
		 */
		TypeTir* getType() const;
};

#endif
```

```
#ifndef TYPE_BONUS_H
#define TYPE_BONUS_H

/**
 * \file TypeBonus.h
 * \brief DÃ©claration de la classe TypeBonus et de l'Ã©numÃ©ration BonusEffect
 * \author Quentin HARSCOÃM-^KT
 */

#include "constantes.h"

/**
 * \struct BonusEffect
 * \brief CatÃ©gories de bonus
 */

enum BonusEffect
{
        POWER, /**< Gain de puissance */
        LIFE, /**< Vie supplÃ©mentaire */
        BOMB, /**< Bombe supplÃ©mentaire */
        HP /**< Gain de points de vie */
};

/**
 * \class TypeBonus TypeBonus.h
 * \brief Classe reprÃ©sentant les diffÃ©rents types de Bonus
 * \author Quentin HARSCOÃM-^KT
 */

class TypeBonus
{
        private:
                BonusEffect my_effect;
                int my_value;
                float my_frequency; // FrÃ©quence/probabilitÃ© d'apparition, comprise entre 0 et 1
                float my_scale; // Ã M-^Ichelle du Bonus (afin d'en avoir des gros et petits avec le mÃªme BonusEffect)

        public:

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param effect : Effet/CatÃ©gorie du Bonus
                 * \param value : Valeur du Bonus (le gain apportÃ©)
                 * \param frequency : FrÃ©quence/probabilitÃ© d'apparition du bonus, comprise entre 0 et 1
                 * \param scale : Taille du Bonus
                 */
                TypeBonus(BonusEffect effect, int value, float frequency, float scale);


                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_effect
                 *
                 * \return : L'effet/catÃ©gorie du Bonus
                 */
                BonusEffect getEffect() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_value
                 *
                 * \return : La valeur du Bonus (le gain apportÃ©)
                 */
                int getValue() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_frequency
                 *
                 * \return : La frÃ©quence/probabilitÃ© d'apparition du Bonus
                 */
                float getFrequency() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_scale
                 *
                 * \return : La taille du Bonus
                 */
                float getScale() const;
};

#endif
```



```
#ifndef TYPE_BOSS_H
#define TYPE_BOSS_H

/**
 * \file TypeBoss.h
 * \brief DÃ©claration de la classe TypeBoss
 * \author Quentin HARSCOÃM-^KT
 */

#include "TypeEnnemi.h"

/**
 * \class TypeBoss TypeBoss.h
 * \brief Classe reprÃ©sentant les diffÃ©rents types de Boss
 * \author Quentin HARSCOÃM-^KT
 */

class TypeBoss : public TypeEnnemi
{
        private:
                float my_changePatternDelay;

        public:

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param score : Points rapportÃ©s par le Boss
                 * \param speed : Vitesse du Boss
                 * \param shootingSpeed : Vitesse de tir du Boss
                 * \param hp : Points de vie du Boss
                 * \param w : Largeur du Boss
                 * \param h : Hauteur du Boss
                 * \param typeTir : Type de tir du Boss
                 * \param shootPattern : Pattern de tir du Boss
                 * \param filename : Nom du fichier du sprite du Boss
                 * \param changePatternDelay : Intervalle de temps entre deux changements de pattern du Boss
                 */
                TypeBoss(int score, int speed, float shootingSpeed, int hp, int w, int h, TypeTir* typeTir, ShootPattern
 shootPattern, std::string filename,
                                        float changePatternDelay);


                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_changePatternDelay
                 *
                 * \return : L'intervalle de temps entre deux changements de pattern du Boss
                 */
                float getChangePatternDelay() const;

};

#endif
```

```
#ifndef TYPE_ENNEMI_H
#define TYPE_ENNEMI_H

/**
 * \file TypeEnnemi.h
 * \brief DÃ©claration de la classe TypeEnnemi et de l'Ã©numÃ©ration ShootPattern
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ«T
 */

#include <string>
#include "TypeTir.h"

/**
 * \struct ShootPattern
 * \brief CatÃ©gories de patterns
 */

enum ShootPattern
{
        SIMPLE, /**< Tir simple en ligne droite */
        M_TRIPLE, /**< Tir triple, "en triangle" */
        TOPLAYER,  /**< Tir dirigÃ© vers le Joueur */
        HALFCIRCLE /**< Tir en demi-cercle, devant le Vaisseau */
};

/*      M-TRIPLE =  S
                        /|\
                       / | \
*/

/**
 * \class TypeEnnemi TypeEnnemi.h
 * \brief Classe reprÃ©sentant les diffÃ©rents types d'Ennemi
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃ«T
 */

class TypeEnnemi
{
        private :
                std::string my_sprite_filename;
                int my_score;
                int my_speed;
                float my_shootingSpeed; // cadence de tir, en secondes (tir 1 fois toutes les x secondes)
                int my_hp;
                int my_h;
                int my_w;
                TypeTir* my_typeTir;
                ShootPattern my_shootPattern;

        public :

                /**
                 * \brief Constructeur par dÃ©faut
                 */
                TypeEnnemi();

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param score : Points rapportÃ©s par l'Ennemi
                 * \param speed : Vitesse de l'Ennemi
                 * \param shootingSpeed : Vitesse de tir de l'Ennemi
                 * \param hp : Points de vie de l'Ennemi
                 * \param w : Largeur de l'Ennemi
                 * \param h : Hauteur de l'Ennemi
                 * \param typeTir : Type de tir de l'Ennemi
                 * \param shootPattern : Pattern de tir de l'Ennemi
                 * \param filename : Nom du fichier du sprite de l'Ennemi
                 */
                TypeEnnemi(int score, int speed, float shootingSpeed, int hp, int w, int h, TypeTir* typeTir, ShootPatte
rn shootPattern, std::string filename);

                /**
                 * \brief Destructeur
                 */
                virtual ~TypeEnnemi();

                //Accesseurs


                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_score
                 *
                 * \return : Les points rapportÃ©s par l'Ennemi
                 */
                int getScore() const;

                /**
                 * \brief Accesseur
```



```
                 *
                 * Accesseur de my_speed
                 *
                 * \return : La vitesse de l'Ennemi
                 */
                int getSpeed() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_shootingSpeed
                 *
                 * \return : La vitesse de tir de l'Ennemi
                 */
                float getShootingSpeed() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_hp
                 *
                 * \return : Les points de vie de l'Ennemi
                 */
                int getHP() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_w
                 *
                 * \return : La largeur de l'Ennemi
                 */
                int getW() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_h
                 *
                 * \return : La hauteur de l'Ennemi
                 */
                int getH() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_filename
                 *
                 * \return : La nom du fichier du sprite de l'Ennemi
                 */
                std::string getFilename() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_typeTir
                 *
                 * \return : Le type de tir de l'Ennemi
                 */
                TypeTir* getTypeTir() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_shootPattern
                 *
                 * \return : Le pattern de tir de l'Ennemi
                 */
                ShootPattern getShootPattern() const;


};

#endif
```

```
#ifndef TYPE_TIR_H
#define TYPE_TIR_H

/**
 * \file TypeTir.h
 * \brief DÃ©claration de la classe TypeTir
 * \author Quentin HARSCOÃM-^KT
 */

#include <string>

/**
 * \class TypeTir TypeTir.h
 * \brief Classe reprÃ©sentant les diffÃ©rents types de Tir
 * \author Quentin HARSCOÃM-^KT
 */

class TypeTir
{
        private:

                int my_w;
                int my_h;
                int my_damages;
                int my_speed;
                std::string my_sprite_filename;

        public :

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param w : Largeur du Tir
                 * \param h : Hauteur du Tir
                 * \param damages : Dommages infligÃ©s par le Tir
                 * \param speed : Vitesse du Tir
                 * \param filename : Nom du fichier du sprite du Tir
                 */
                TypeTir(int w, int h, int damages, int speed, std::string filename);


                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_filename
                 *
                 * \return : Le nom du fichier du sprite du Tir
                 */
                std::string getFilename() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_h
                 *
                 * \return : La hauteur du Tir
                 */
                int getH() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_w
                 *
                 * \return : La largeur du Tir
                 */
                int getW() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_w
                 *
                 * \return : Les dommages infligÃ©s par le Tir
                 */
                int getDamages() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_w
                 *
                 * \return : La vitesse du Tir
                 */
                int getSpeed() const;
};

#endif
```



```
#ifndef VAISSEAU_H
#define VAISSEAU_H

/**
 * \file Vaisseau.h
 * \brief DÃ©claration de la classe Vaisseau
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

#include <SFML/Window.hpp>
#include "MovableElement.h"
#include "Tir.h"
#include <vector>

class GameModel;
class TypeTir;

/**
 * \class Vaisseau Vaisseau.h
 * \brief Classe reprÃ©sentant les vaisseaux
 * \author ThÃ©o CHASSAIGNE
 * \author Quentin HARSCOÃM-^KT
 */

class Vaisseau: public MovableElement
{
    private:
        int my_hp;
        int my_pvMax;
        bool my_isAlive;

        Vaisseau();

    protected:
        TypeTir* my_typeTir; /**< Pointeur vers le type des tirs du Vaisseau */
        sf::Clock my_clock; /**< Pendule interne au Vaisseau, pour y gÃ©rer le temps */
        float my_shootingSpeed; /**< Cadence de tir, en secondes (un tir toutes les x secondes) */

    public:

                /**
                 * \brief Constructeur paramÃ©trÃ©
                 *
                 * \param hp : Points de vie initiaux du Vaisseau
                 * \param type : Type de tir initial du Vaisseau
                 * \param shootingSpeed : Cadence de tir initiale du Vaisseau
                 */
        Vaisseau(int hp, TypeTir* type, float shootingSpeed);

                /**
                 * \brief Destructeur
                 */
        virtual ~Vaisseau();


//Accesseurs

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_hp
                 *
                 * \return : Les points de vie actuels du Vaisseau
                 */
        int getHP() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_pvMax
                 *
                 * \return : Les points de vie maximum du Vaisseau
                 */
        int getMaxHP() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_typeTir
                 *
                 * \return : Le type de tir actuel du Vaisseau
                 */
        TypeTir* getTypeTir() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_isAlive
                 *
                 * \return : L'Ã©tat actuel du Vaisseau (\b true si le Vaisseau n'est pas dÃ©truit, \b false sinon)
                 */
        bool getAlive() const;
```

```
                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_shootingSpeed
                 *
                 * \return : La cadence de tir actuelle du Vaisseau
                 */
        float getShootingSpeed() const;

                /**
                 * \brief Accesseur
                 *
                 * Accesseur de my_clock
                 *
                 * \return : La Clock interne au Vaisseau
                 */
        sf::Clock* getClock();


                /**
                 * \brief DÃ©finir les points de vie du Vaisseau
                 *
                 * \param i : Nouveaux points de vie du Vaisseau
                 */
        void setHP(int i);

                /**
                 * \brief DÃ©finir le type de tir du Vaisseau
                 *
                 * \param type : Nouveau type de tir du Vaisseau
                 */
        void setTypeTir(TypeTir* type);

                /**
                 * \brief DÃ©finir l'Ã©tat du Vaisseau
                 *
                 * \param b : Nouvel Ã©tat du Vaisseau (\b true si le Vaisseau n'est pas dÃ©truit, \b false sinon)
                 */
        void setAlive(bool b);

//MÃ©thodes

                /**
                 * \brief Faire tirer le Vaisseau (mÃ©thode virtuelle pure)
                 * \param model : ModÃ¨le du jeu
                 */
        virtual void shoot(GameModel* model) = 0;

                /**
                 * \brief Indiquer le tir touchant le Vaisseau
                 *
                 * \param model : ModÃ©le du jeu (mÃ©thode virtuelle pure)
                 * \return : Le tir touchant le vaisseau contrÃ´lÃ©
                 */
        virtual std::vector<Tir*>::iterator isShot(GameModel* model) const = 0;
};

#endif
```

C Documentation

Également disponible en HTML dans le dossier doc.

Space Sheep Xtreme Turbo Penguin Edition

Généré par Doxygen 1.8.4

Dimanche Juin 2 2013 19 :39 :59

Table des matières

Chapitre 1

Liste des bogues

Membre GameView : :synchronize ()

Une technique plus optimisée a été trouvée pour cette méthode, mais elle entraîne parfois des erreurs de segmentation, pour une raison inconnue

Membre GameView : :viewHighscores (bool &quit)

Pour une raison inconnue, un Event : :Closed se produit à l'ouverture du menu. Pour pallier à ça, il a fallu empêcher de fermer le programme "normalement" une fois sur cet écran (l'utilisation de la touche Escape est toujours possible cependant)

Chapitre 2

Index hiérarchique

2.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Chapitre 3

Index des classes

3.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 4

Index des fichiers

4.1 Liste des fichiers

Liste de tous les fichiers documentés avec une brève description :

Chapitre 5

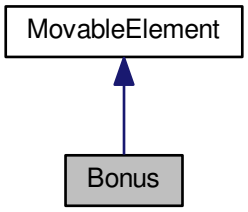
Documentation des classes

5.1 Référence de la classe Bonus

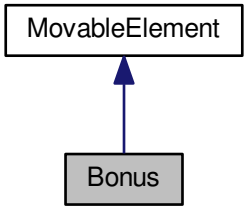
Classe représentant les bonii à ramasser.

`#include <Bonus.h>`

Graphe d'héritage de Bonus :



Graphe de collaboration de Bonus :



Fonctions membres publiques

– Bonus (int x, int y, TypeBonus *type)
Constructeur paramétré
– TypeBonus * getType () const
Accesseur.

Additional Inherited Members

5.1.1 Description détaillée

Classe représentant les bonii à ramasser.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.1.2 Documentation des constructeurs et destructeur

5.1.2.1 Bonus : :Bonus (int *x*, int *y*, TypeBonus * *type*)

Constructeur paramétré

Paramètres

x	: Abscisse du bonus
y	: Ordonnée du bonus
type	: Type du bonus

5.1.3 Documentation des fonctions membres

5.1.3.1 TypeBonus * Bonus : :getType ( ) const

Accesseur.

Accesseur de my_type

Renvoie

: Le type du Bonus

La documentation de cette classe a été générée à partir des fichiers suivants :
– Bonus.h
– Bonus.cc

5.2 Référence de la structure BonusEffect

Catégories de bonus.

`#include <TypeBonus.h>`

5.2.1 Description détaillée

Catégories de bonus.

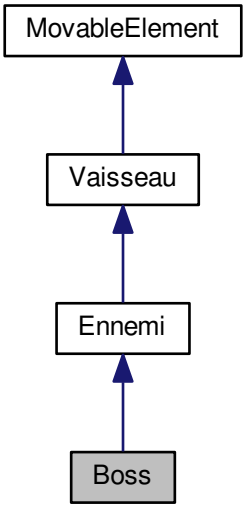
La documentation de cette structure a été générée à partir du fichier suivant :
– TypeBonus.h

5.3 Référence de la classe Boss

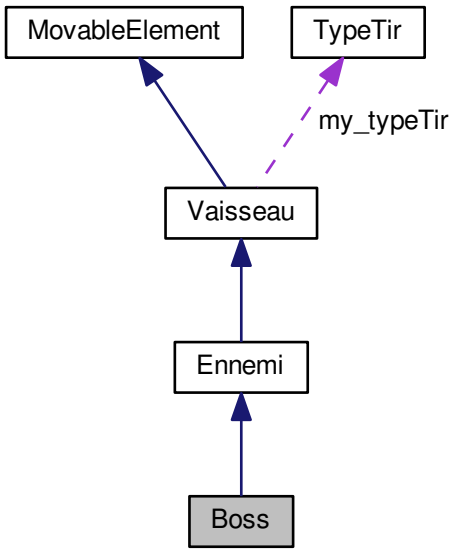
Classe représentant les boss de fin de niveau.

`#include <Boss.h>`

Graphe d'héritage de Boss :



Graphe de collaboration de Boss :



Fonctions membres publiques

- Boss (TypeBoss ∗type)
  Constructeur paramétré
- ShootPattern getShootPattern () const
  Accesseur.
- void changePattern ()
  Changer le pattern.
- void move ()
  Déplacer le boss.

Additional Inherited Members

5.3.1 Description détaillée

Classe représentant les boss de fin de niveau.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.3.2 Documentation des constructeurs et destructeur

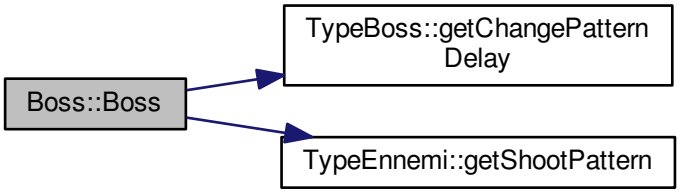
5.3.2.1 Boss : :Boss (TypeBoss ∗ *type*)

Constructeur paramétré

Paramètres

type	: Type du boss

Voici le graphe d'appel pour cette fonction :



5.3.3 Documentation des fonctions membres

5.3.3.1 void Boss : :changePattern ( )

Changer le pattern.

Méthode permettant de changer le pattern actuellement utilisé

5.3.3.2 ShootPattern Boss : :getShootPattern ( ) const [virtual]

Accesseur.

Accesseur de my_actualShootPattern

Renvoie

: Le pattern actuellement utilisé

Réimplémentée à partir de Ennemi.

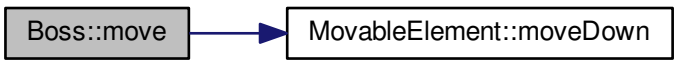
5.3.3.3 void Boss : :move ( ) [virtual]

Déplacer le boss.

Méthode permettant de déplacer le Boss

Réimplémentée à partir de MovableElement.

Voici le graphe d'appel pour cette fonction :



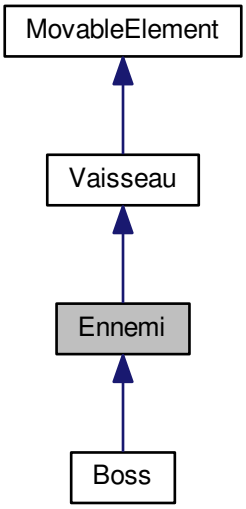
La documentation de cette classe a été générée à partir des fichiers suivants :
– Boss.h
– Boss.cc

5.4 Référence de la classe Ennemi

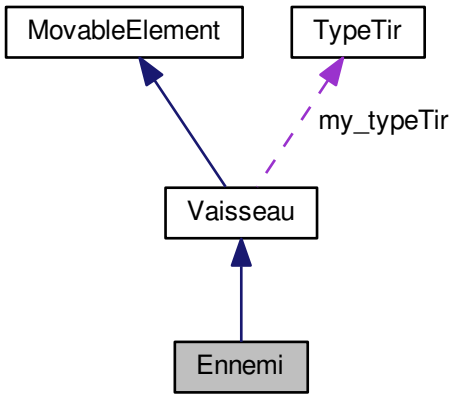
Classe représentant les ennemis du joueur.

`#include <Ennemi.h>`

Graphe d'héritage de Ennemi :



Graphe de collaboration de Ennemi :



Fonctions membres publiques

- Ennemi ()
  Constructeur par défaut.
- Ennemi (TypeEnnemi *type)
  Constructeur paramétré
- virtual ~Ennemi ()
  Destructeur.
- int getScore () const

Accesseur.
- TypeEnnemi ∗ getType () const
  Accesseur.
- virtual ShootPattern getShootPattern () const
  Accesseur.
- void shoot (GameModel ∗model)
  Faire tirer l'ennemi.
- std : :vector< Tir ∗ > : :iterator isShot (GameModel ∗model) const
  Indiquer le tir touchant l'ennemi.
- void move ()
  Déplacer l'ennemi.

Additional Inherited Members

5.4.1 Description détaillée

Classe représentant les ennemis du joueur.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.4.2 Documentation des constructeurs et destructeur

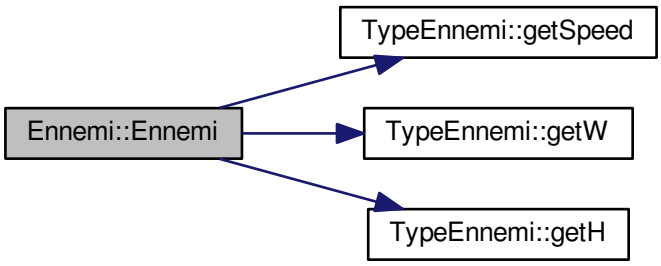
5.4.2.1 Ennemi : :Ennemi (TypeEnnemi ∗ *type*)

Constructeur paramétré

Paramètres

type	: Type d'ennemi

Voici le graphe d'appel pour cette fonction :



5.4.3 Documentation des fonctions membres

5.4.3.1 int Ennemi : :getScore ( ) const

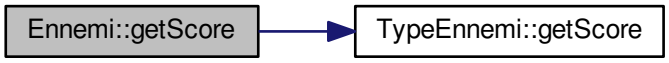
Accesseur.

Accesseur de my_type.my_score

Renvoie

: Les points rapportés par l'ennemi

Voici le graphe d'appel pour cette fonction :



5.4.3.2 ShootPattern Ennemi : :getShootPattern ( ) const [virtual]

Accesseur.

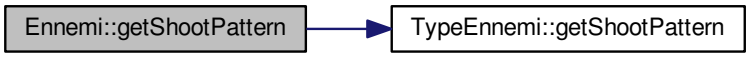
Accesseur de my_type.my_shootPattern

Renvoie

: Le pattern utilisé par l'ennemi

Réimplémentée dans Boss.

Voici le graphe d'appel pour cette fonction :



5.4.3.3 TypeEnnemi * Ennemi : :getType ( ) const

Accesseur.

Accesseur de my_type

Renvoie

: Le type d'ennemi

5.4.3.4 vector< Tir * > : :iterator Ennemi : :isShot (GameModel * *model*) const [virtual]

Indiquer le tir touchant l'ennemi.

Paramètres

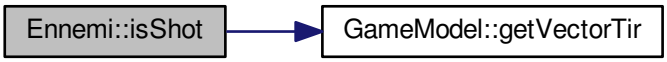
model	: Modéle du jeu

Renvoie

: Le tir touchant l'ennemi

Implémente Vaisseau.

Voici le graphe d'appel pour cette fonction :



5.4.3.5 void Ennemi : :shoot (GameModel * *model*) [virtual]

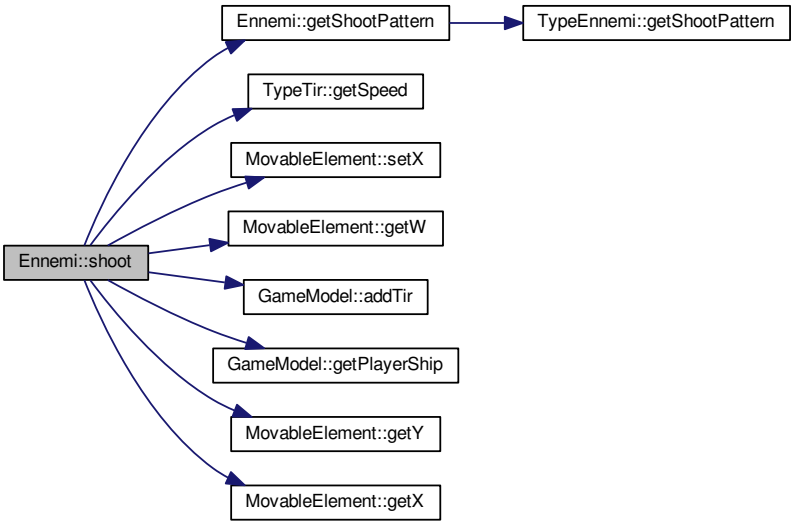
Faire tirer l'ennemi.

Paramètres

model	: Modéle du jeu

Implémente Vaisseau.

Voici le graphe d'appel pour cette fonction :



La documentation de cette classe a été générée à partir des fichiers suivants :
– Ennemi.h
– Ennemi.cc

5.5 Référence de la classe GameModel

Classe représentant le Modèle du jeu.

`#include <GameModel.h>`

Fonctions membres publiques

- GameModel (int viesMax, int initialLives, int difficulty)
  Constructeur paramétré
- ~GameModel ()
  Destructeur.
- void nextStep ()
  Passer à "l'étape" suivante.
- void addEnnemy (TypeEnnemi *type)
  Ajouter un Ennemi.
- void addTir (Tir *t)
  Ajouter un Tir.
- void addBonus (int x, int y)
  Ajouter un Bonus.
- void removeEnnemy (Ennemi *ennemi)
  Supprimer un Ennemi.
- void removeEnnemy (std : :vector< Ennemi * > : :iterator it)
  Supprimer un Ennemi.
- void saveScore (string name)
  Sauvegarder le score.
- void changeNiveau ()
  Passer au Niveau suivant.
- void gameOver ()
  Tuer le Joueur.
- void getShipPos (int &x, int &y) const
  Accesseur.
- bool getNextLevel () const
  Accesseur.
- int getLevelID () const
  Accesseur.
- int getScore () const
  Accesseur.
- int getDifficulty () const
  Accesseur.
- string getStringLevelID () const
  Accesseur.
- string getStringScore () const
  Accesseur.
- TypeEnnemi *const * getEnnemyTypes () const
  Accesseur.
- TypeTir *const * getTirTypes () const
  Accesseur.
- TypeBoss * getBossType () const
  Accesseur.
- Joueur * getPlayerShip () const
  Accesseur.
- vector< Ennemi * > * getVectorEnnemis ()
  Accesseur.
- vector< Tir * > * getVectorTir ()
  Accesseur.
- const vector< Bonus * > & getVectorBonuses ()
  Accesseur.
- vector< MovableElement * > * getDeletedItems ()
  Accesseur.
- Horloge * getClock ()
  Accesseur.
- void setScore (int score)
  Définir le score.
- void setNextLevel (bool nextLvl)
  Définir si l'on passe au niveau suivant.

5.5.1 Description détaillée

Classe représentant le Modèle du jeu.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.5.2 Documentation des constructeurs et destructeur

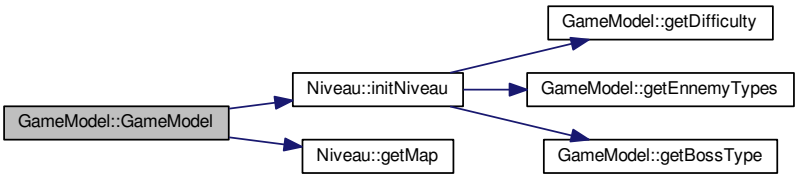
5.5.2.1 GameModel : :GameModel (int *viesMax,* int *initialLives,* int *difficulty*)

Constructeur paramétré

Paramètres

viesMax	: Nombre maximum de vies du Joueur
initialLives	: Nombre initial de vies du Joueur
difficulty	: Difficulté du jeu

Voici le graphe d'appel pour cette fonction :



5.5.3 Documentation des fonctions membres

5.5.3.1 void GameModel : :addBonus (int *x,* int *y*)

Ajouter un Bonus.

Paramètres

x	: Abscisse du Bonus à ajouter
y	: Ordonnée du Bonus à ajouter

5.5.3.2 void GameModel : :addEnnemy (TypeEnnemi * *type*)

Ajouter un Ennemi.

Paramètres

type	: Type de l'Ennemi à ajouter

5.5.3.3 void GameModel : :addTir (Tir * *t*)

Ajouter un Tir.

Paramètres

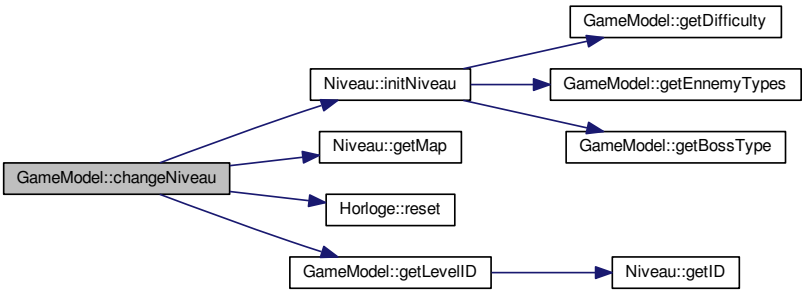
t	: Tir à ajouter

5.5.3.4 void GameModel : :changeNiveau ( )

Passer au Niveau suivant.

Vide le vector des Tir, réinitialise la Clock et redéfinit le Boss

Voici le graphe d'appel pour cette fonction :

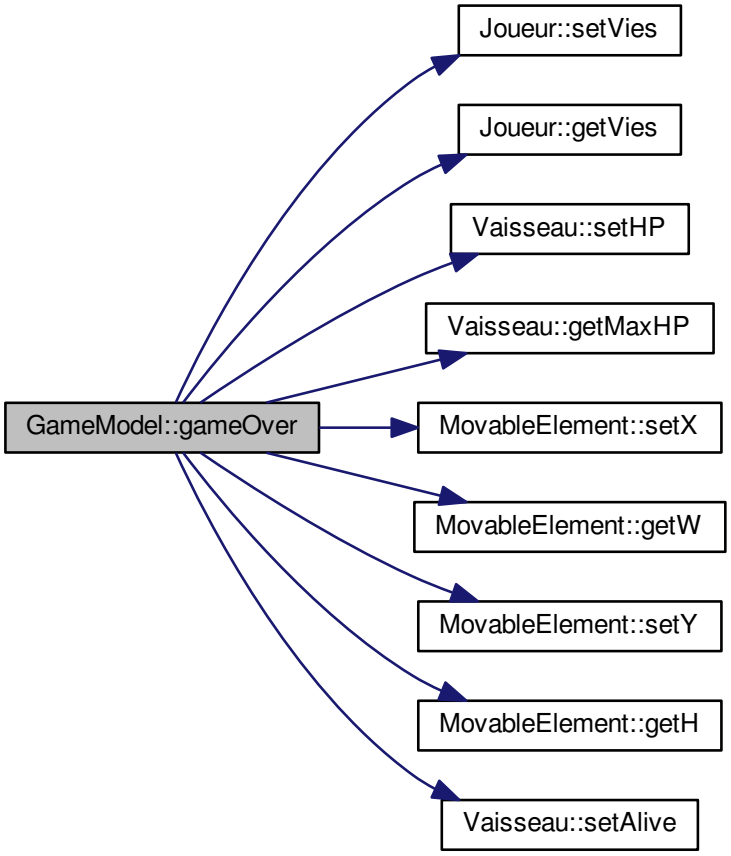


5.5.3.5 void GameModel : :gameOver ( )

Tuer le Joueur.

Méthode permettant de gérer la mort du Joueur

Voici le graphe d'appel pour cette fonction :



5.5.3.6 TypeBoss * GameModel : :getBossType ( ) const

Accesseur.

Accesseur de my_boss_types

Renvoie

: Le tableau contenant les différents TypeBoss

5.5.3.7 Horloge * GameModel : :getClock ( )

Accesseur.

Accesseur de my_clock

Renvoie

: L'Horloge du jeu

5.5.3.8 vector< MovableElement * > * GameModel : :getDeletedItems ( )

Accesseur.

Accesseur de my_tirs

Renvoie

: Le vecteur contenant les objets à supprimer

5.5.3.9 int GameModel : :getDifficulty ( ) const

Accesseur.

Accesseur de my_difficulty

Renvoie

: La difficulté du jeu

5.5.3.10 TypeEnnemi *const * GameModel : :getEnnemyTypes ( ) const

Accesseur.

Accesseur de my_ennemi_types

Renvoie

: Le tableau contenant les différents TypeEnnemi

5.5.3.11 int GameModel : :getLevelID ( ) const

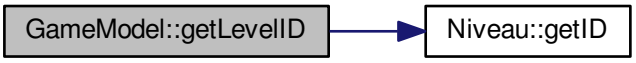
Accesseur.

Accesseur de niveau.my_id

Renvoie

: L'identifiant du Niveau en cours

Voici le graphe d'appel pour cette fonction :



5.5.3.12 bool GameModel : :getNextLevel ( ) const

Accesseur.

Accesseur de my_nextLevel

Renvoie

: Si l'on passe ou non au niveau suivant (**true** si oui, **sinon**)

5.5.3.13 Joueur * GameModel : :getPlayerShip ( ) const

Accesseur.

Accesseur de my_ship

Renvoie

: Le Joueur

5.5.3.14 int GameModel : :getScore ( ) const

Accesseur.

Accesseur de my_score

Renvoie

: Le score actuel

5.5.3.15 void GameModel : :getShipPos (int & *x*, int & *y*) const

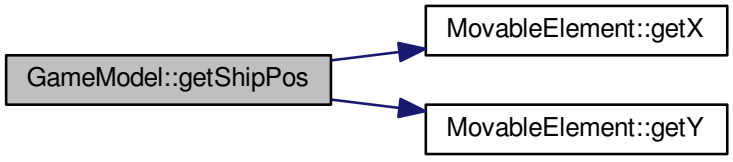
Accesseur.

Méthode récupérant la position du Joueur

Paramètres

x	: (Sortie seulement) Abscisse du Joueur
y	: (Sortie seulement) Ordonnée du Joueur

Voici le graphe d'appel pour cette fonction :



5.5.3.16 string GameModel : :getStringLevelID ( ) const

Accesseur.

Accesseur de niveau.my_id

Renvoie

: L'identifiant du Niveau en cours, sous forme de chaîne de caractères

Voici le graphe d'appel pour cette fonction :

5.5.3.17 string GameModel : :getStringScore ( ) const

Accesseur.

Accesseur de my_score

Renvoie

: Le score actuel, sous forme de chaîne de caractères

5.5.3.18 TypeTir ∗const ∗ GameModel : :getTirTypes ( ) const

Accesseur.

Accesseur de my_tirs_types

Renvoie

: Le tableau contenant les différents TypeTir

5.5.3.19 const vector< Bonus ∗ > & GameModel : :getVectorBonuses ( )

Accesseur.

Accesseur de my_bonuses

Renvoie

: Le vecteur contenant les Bonus présents

5.5.3.20 vector< Ennemi ∗ > ∗ GameModel : :getVectorEnnemis ( )

Accesseur.

Accesseur de my_ennemis

Renvoie

: Le vecteur contenant les Ennemi présents

5.5.3.21 vector< Tir * > * GameModel : :getVectorTir ( )

Accesseur.

Accesseur de my_tirs

Renvoie

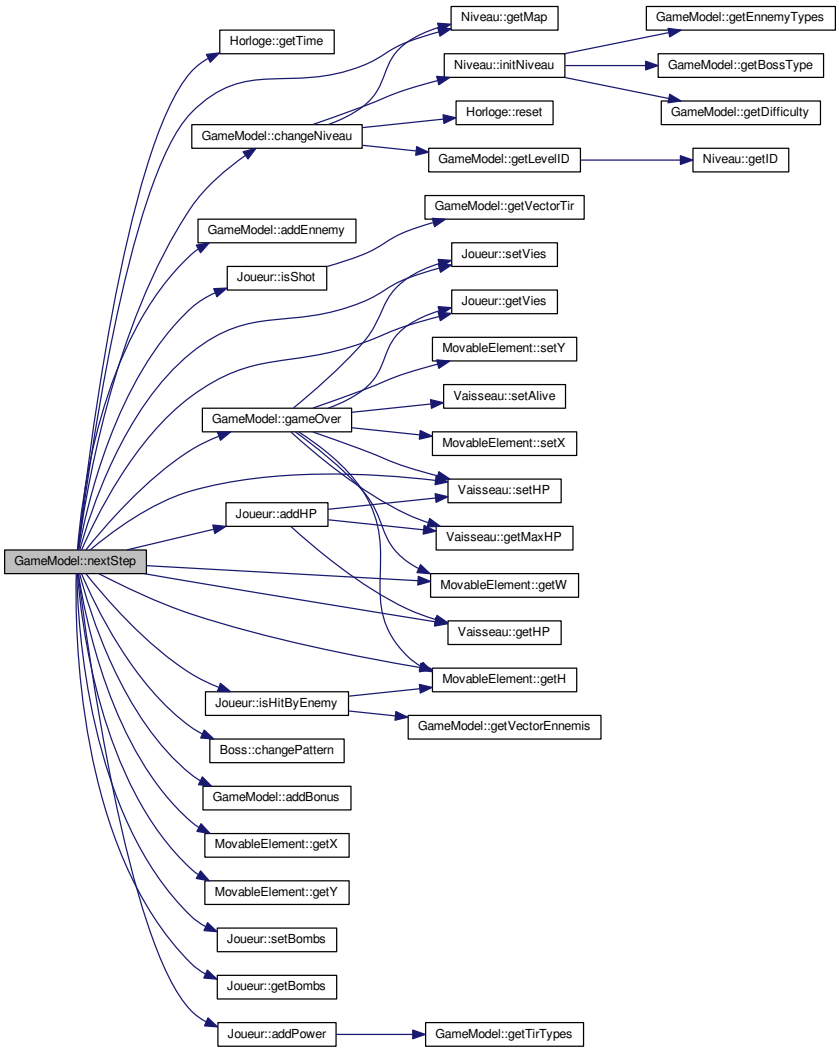
: Le vecteur contenant les Tir présents

5.5.3.22 void GameModel : :nextStep ( )

Passer à "l'étape" suivante.

Méthode permettant de faire progresser le jeu

Voici le graphe d'appel pour cette fonction :



5.5.3.23 void GameModel : :removeEnnemy (Ennemi * *ennemi*)

Supprimer un Ennemi.

Paramètres

ennemi	: Adresse de l'Ennemi à supprimer

5.5.3.24 void GameModel : :removeEnnemy (std : :vector< Ennemi * > : :iterator *it*)

Supprimer un Ennemi.

Paramètres

it	: Itérateur vers l'Ennemi à supprimer

5.5.3.25 void GameModel : :saveScore (string *name*)

Sauvegarder le score.

Paramètres

name	: Pseudo à afficher

5.5.3.26 void GameModel : :setNextLevel (bool *nextLvl*)

Définir si l'on passe au niveau suivant.

Paramètres

nextLvl	: **true** si l'on passe au niveau suivant, **false** sinon

5.5.3.27 void GameModel : :setScore (int *score*)

Définir le score.

Paramètres

score	: Nouveau score

La documentation de cette classe a été générée à partir des fichiers suivants :
- GameModel.h
- GameModel.cc

5.6 Référence de la classe GameView

Classe représentant la Vue du jeu.

`#include <GameView.h>`

Fonctions membres publiques

- GameView (int w, int h, int bpp)
  Constructeur paramétré
- ~GameView ()
  Destructeur.
- void setModel (GameModel *model)
  Définir le Modèle.
- void updateLanguage ()
  Mettre à jour la langue.

- void updateVolume ()

  Mettre à jour le volume sonore.

- void updateSound ()

  Mettre à jour les sons utilisés.

- void bgscroll ()

  Faire défiler l'arrière-plan.

- void explosion (GraphicElement &elem, Ennemi ∗adr)
  Dessiner une explosion.
- void draw ()

  Dessiner tous les éléments dans la Vue.

- void Display ()

  Afficher les éléments dans une fenêtre.

- void transition (std : :string text, float t)
  Dessiner une transition.
- void synchronize ()
  Synchroniser la Vue avec le Modèle.
- bool treatEvents (bool &quit)
  Traite les événements de la SFML.
- int pause ()

  Dessiner l'écran de pause.

- void intro ()

  Dessiner l'introduction du jeu.

- int menu ()

  Dessiner le menu principal.

- void options (bool &quit, int &difficulty, int &initialLives)
  Dessiner le menu des options.
- void viewHighscores (bool &quit)
  Dessiner le menu des meilleurs scores.
- void askName ()

  Dessiner la demande de son pseudo à l'utilisateur.

5.6.1 Description détaillée

Classe représentant la Vue du jeu.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.6.2 Documentation des constructeurs et destructeur

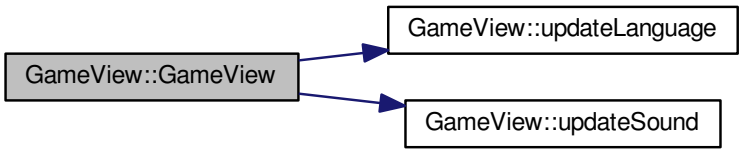
5.6.2.1 GameView : :GameView (int *w,* int *h,* int *bpp*)

Constructeur paramétré

Paramètres

w	: Largeur de la Vue
h	: Hauteur de la Vue
bpp	: Bits par pixel (profondeur des couleurs)

Voici le graphe d'appel pour cette fonction :



5.6.3 Documentation des fonctions membres

5.6.3.1 void GameView : :explosion (GraphicElement & *elem,* Ennemi * *adr*)

Dessiner une explosion.

Paramètres

elem	: Sprite de l'Ennemi explosant
adr	: Ennemi explosant

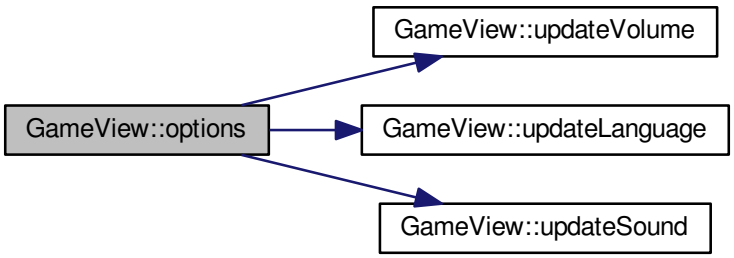
5.6.3.2 void GameView : :options (bool & *quit,* int & *difficulty,* int & *initialLives*)

Dessiner le menu des options.

Paramètres

quit	: (Sortie seulement) État du programme
difficulty	: (Sortie seulement) Difficulté du jeu
initialLives	: (Sortie seulement) Nombre initial de vies du Joueur

Voici le graphe d'appel pour cette fonction :



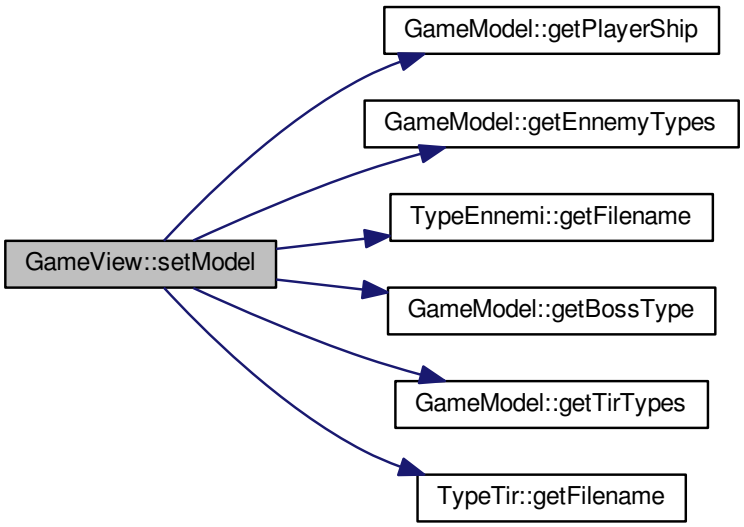
5.6.3.3 void GameView : :setModel (GameModel * *model*)

Définir le Modèle.

Paramètres

model	: GameModel à utiliser

Voici le graphe d'appel pour cette fonction :

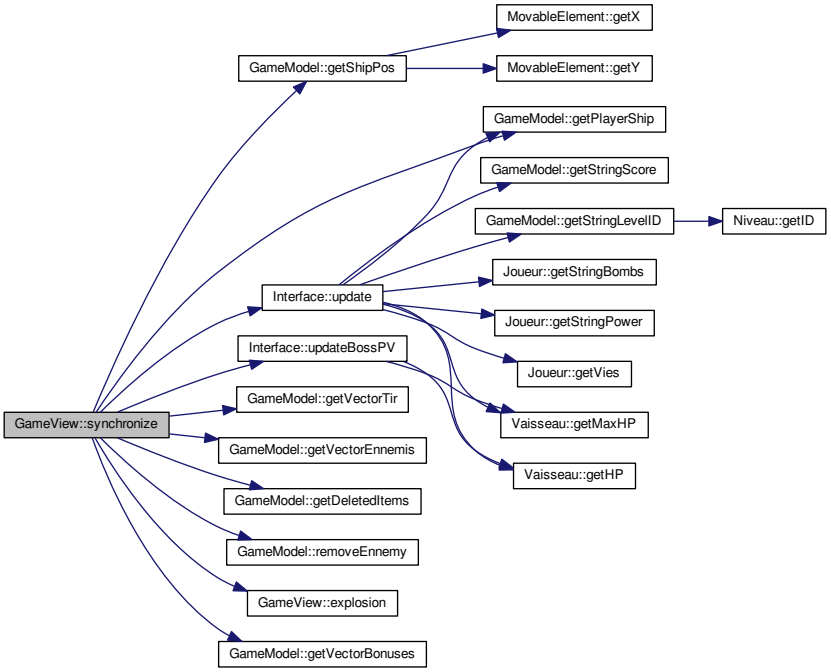


5.6.3.4 void GameView : :synchronize ( )

Synchroniser la Vue avec le Modèle.

Bogue Une technique plus optimisée a été trouvée pour cette méthode, mais elle entraîne parfois des erreurs de segmentation, pour une raison inconnue

Voici le graphe d'appel pour cette fonction :



5.6.3.5 void GameView : :transition (std : :string *text*, float *t*)

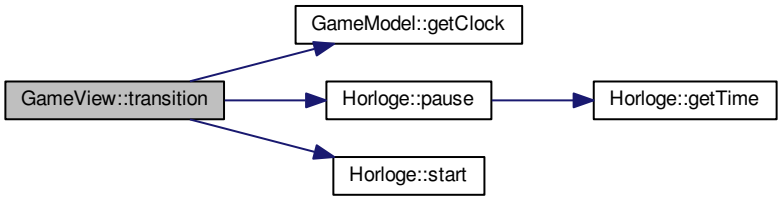
Dessiner une transition.

Méthode permettant d'afficher un texte et un compte à rebours après la mort du Joueur ou avant le début d'un Niveau par exemple

Paramètres

text	: Texte de la transition
t	: Durée de la transition (en secondes)

Voici le graphe d'appel pour cette fonction :



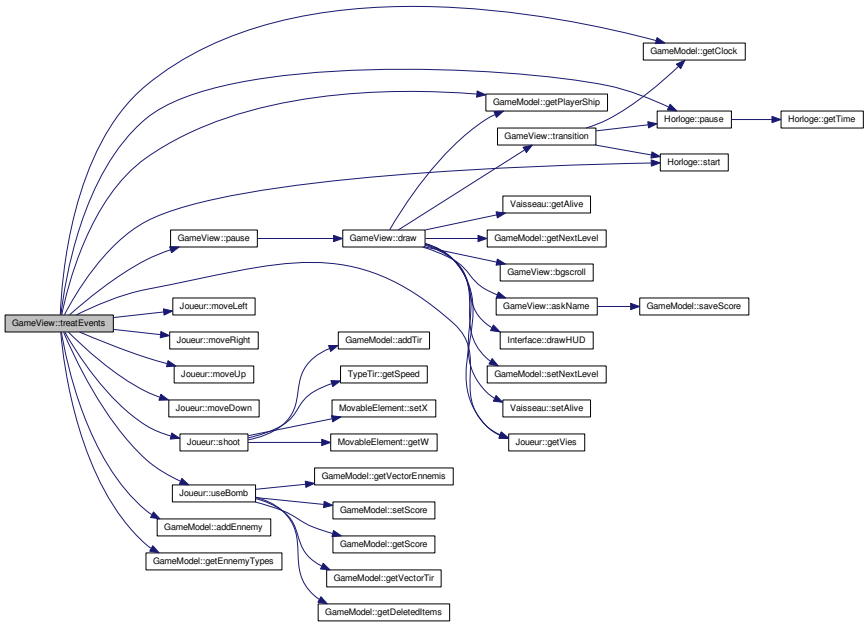
5.6.3.6 bool GameView : :treatEvents (bool & *quit*)

Traite les événements de la SFML.

Paramètres

quit	: (Sortie seulement) État du programme

Voici le graphe d'appel pour cette fonction :



5.6.3.7 void GameView : :viewHighscores (bool & *quit*)

Dessiner le menu des meilleurs scores.

Paramètres

quit	: (Sortie seulement) État du programme

Bogue Pour une raison inconnue, un Event : :Closed se produit à l'ouverture du menu. Pour pallier à ça, il a fallu empêcher de fermer le programme "normalement" une fois sur cet écran (l'utilisation de la touche Escape est toujours possible cependant)

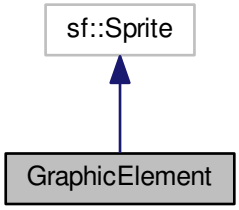
La documentation de cette classe a été générée à partir des fichiers suivants :
– GameView.h
– GameView.cc

5.7 Référence de la classe GraphicElement

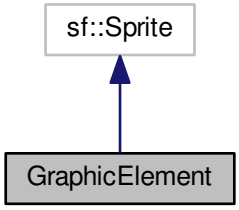
Classe représentant les éléments à afficher.

`#include <GraphicElement.h>`

Graphe d'héritage de GraphicElement :



Graphe de collaboration de GraphicElement :



Fonctions membres publiques

- GraphicElement ()
  Constructeur par défaut.
- GraphicElement (sf : :Image ∗image, int x, int y, int w, int h)
  Constructeur paramétré
- virtual ∼GraphicElement ()
  Destructeur.
- virtual void draw (sf : :RenderWindow ∗_window)
  Dessiner l'élément à afficher.
- void setPosition (const sf : :Vector2f &pos)
  Modifier la position de l'élément à afficher.
- void setPosition (int x, int y)
  Modifier la position de l'élément à afficher.
- void resize (int w, int h)
  Modifier les dimensions de l'élément à afficher.
- bool getVisible () const
  Accesseur.
- void setVisible (bool visible)
  Définir la visibilité de l'élément à afficher.

5.7.1 Description détaillée

Classe représentant les éléments à afficher.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.7.2 Documentation des constructeurs et destructeur

5.7.2.1 GraphicElement : :GraphicElement (sf : :Image * *image*, int *x*, int *y*, int *w*, int *h*)

Constructeur paramétré

Paramètres

image	: Image à charger
x	: Abscisse de l'emplacement de l'élément à afficher dans la Vue
y	: Ordonnée de l'emplacement de l'élément à afficher dans la Vue
w	: Largeur de l'image à charger
h	: Hauteur de l'image à charger

5.7.3 Documentation des fonctions membres

5.7.3.1 void GraphicElement : :draw (sf : :RenderWindow * *_window*) [virtual]

Dessiner l'élément à afficher.

Méthode permettant de dessiner l'élément à afficher dans une fenêtre

Paramètres

_window	: Fenêtre où dessiner l'élément à afficher

5.7.3.2 bool GraphicElement : :getVisible ( ) const

Accesseur.

Accesseur de _visible

Renvoie

: La visibilité actuelle de l'élément

5.7.3.3 void GraphicElement : :resize (int *w*, int *h*)

Modifier les dimensions de l'élément à afficher.

Paramètres

w	: Nouvelle largeur de l'élément à afficher
h	: Nouvelle hauteur de l'élément à afficher

5.7.3.4 void GraphicElement : :setPosition (const sf : :Vector2f & *pos*)

Modifier la position de l'élément à afficher.

Paramètres

pos	: Vecteur contenant les nouvelles coordonnées de l'élément à afficher

5.7.3.5 void GraphicElement : :setPosition (int *x*, int *y*)

Modifier la position de l'élément à afficher.

Paramètres

x	: Nouvelle abscisse de l'élément à afficher
y	: Nouvelle ordonnée de l'élément à afficher

5.7.3.6 void GraphicElement : :setVisible (bool *visible*)

Définir la visibilité de l'élément à afficher.

Paramètres

visible	: Nouvelle visibilité de l'élément à afficher

La documentation de cette classe a été générée à partir des fichiers suivants :
– GraphicElement.h
– GraphicElement.cc

5.8 Référence de la classe Horloge

Classe représentant le temps dans le jeu.

`#include <Horloge.h>`

Fonctions membres publiques

– Horloge ()
  Constructeur par défaut.
– float getTime ()
  Retourner le temps écoulé
– bool getStatus () const
  Accesseur.
– void start ()
  Démarrer l'horloge.
– void pause ()
  Mettre l'horloge en pause.
– void stop ()
  Arrêter l'horloge.
– void reset ()
  Réinitialiser l'horloge.

5.8.1 Description détaillée

Classe représentant le temps dans le jeu.

Remplace la classe Clock de la SFML 1.6, afin notamment d'implémenter une pause

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.8.2 Documentation des fonctions membres

5.8.2.1 bool Horloge : :getStatus ( ) const

Accesseur.

Accesseur de m_paused

Renvoie

: L'état de l'horloge (**true** si l'horloge est en pause, **false** sinon)

5.8.2.2 float Horloge : :getTime ( )

Retourner le temps écoulé

Renvoie

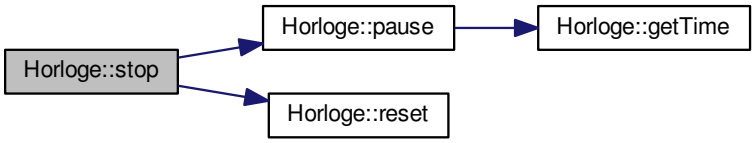
: Le temps écoulé depuis le démarrage de l'horloge

5.8.2.3 void Horloge : :stop ( )

Arrêter l'horloge.

Met l'horloge en pause et la réinitialise

Voici le graphe d'appel pour cette fonction :



La documentation de cette classe a été générée à partir des fichiers suivants :
– Horloge.h
– Horloge.cc

5.9 Référence de la classe Interface

Classe représentant le HUD (affichage tête haute)

`#include <Interface.h>`

Fonctions membres publiques

– Interface ()
  Constructeur par défaut.
– ∼Interface ()
  Destructeur.
– void drawHUD (sf : :RenderWindow ∗_window)

Dessiner le HUD.
– void update (GameModel ∗model, string strings[])
Mettre à jour le HUD.
– void updateBossPV (Boss ∗boss)
Mettre à jour la barre de vie du boss.

5.9.1 Description détaillée

Classe représentant le HUD (affichage tête haute)

Auteur

Quentin HARSCOËT

5.9.2 Documentation des fonctions membres

5.9.2.1 void Interface : :drawHUD (sf : :RenderWindow ∗ *_window*)

Dessiner le HUD.

Méthode permettant de dessiner le HUD dans une fenêtre

Paramètres

_window	: Fenêtre où dessiner le HUD

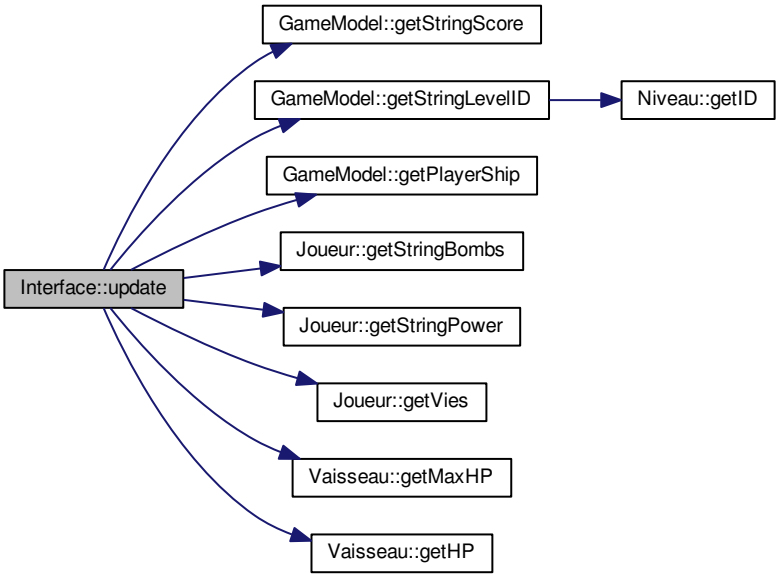
5.9.2.2 void Interface : :update (GameModel ∗ *model,* string *strings[]*)

Mettre à jour le HUD.

Paramètres

model	: Modèle du jeu
strings[]	: Tableau contenant les textes du HUD (utilisé pour la traduction)

Voici le graphe d'appel pour cette fonction :



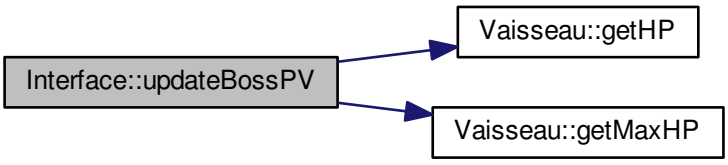
5.9.2.3 void Interface : :updateBossPV (Boss ∗ *boss*)

Mettre à jour la barre de vie du boss.

Paramètres

boss	: Boss affronté

Voici le graphe d'appel pour cette fonction :



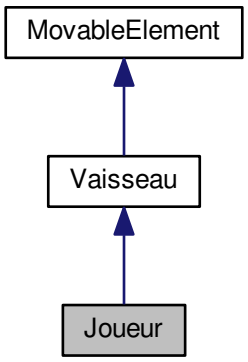
La documentation de cette classe a été générée à partir des fichiers suivants :
– Interface.h
– Interface.cc

5.10 Référence de la classe Joueur

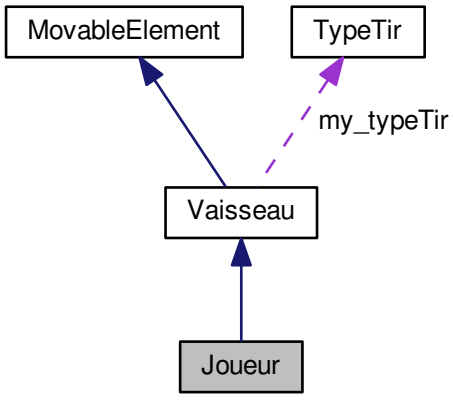
Classe représentant le vaisseau contrôlé

`#include <Joueur.h>`

Graphe d'héritage de Joueur :



Graphe de collaboration de Joueur :



Fonctions membres publiques

- Joueur (int power, TypeTir *typeTir, int vies, int pvMax, int bombs, float shootingSpeed)
  Constructeur paramétré
- void moveLeft ()
  Déplacer le vaisseau contrôlé vers la gauche.
- void moveRight ()
  Déplacer le vaisseau contrôlé vers la droite.

- void moveDown ()
  Déplacer le vaisseau contrôlé vers le bas.
- void moveUp ()
  Déplacer le vaisseau contrôlé vers le haut.
- void shoot (GameModel *model)
  Faire tirer le vaisseau contrôlé
- std : :vector< Tir * > : :iterator isShot (GameModel *model) const
  Indiquer le tir touchant le vaisseau.
- bool isHitByEnemy (GameModel *model)
  Indiquer si le vaisseau contrôlé est touché
- void useBomb (GameModel *model)
  Utiliser une bombe.
- void addPower (int power, GameModel *model)
  Ajouter de la puissance.
- void addHP (int hp)
  Ajouter des points de vie.
- int getVies () const
  Accesseur.
- int getPower () const
  Accesseur.
- int getBombs () const
  Accesseur.
- int getNumberOfShoots () const
  Accesseur.
- std : :string getStringBombs () const
  Accesseur.
- std : :string getStringPower () const
  Accesseur.
- void setVies (int vies)
  Définir le nombre de vies du vaisseau contrôlé
- void setBombs (int bombs)
  Définir la quantité de bombes du vaisseau contrôlé
- void setPower (int i)
  Définir la puissance du vaisseau contrôlé
- void setNumberOfShoots (int n)
  Définir le nombre de tirs simultanés du vaisseau contrôlé

Additional Inherited Members

5.10.1 Description détaillée

Classe représentant le vaisseau contrôlé

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.10.2 Documentation des constructeurs et destructeur

5.10.2.1 Joueur : :Joueur (int *power,* TypeTir * *typeTir,* int *vies,* int *pvMax,* int *bombs,* float *shootingSpeed*)

Constructeur paramétré

Paramètres

power	: Puissance initiale du vaisseau contrôlé
typeTir	: Type de tir initial du vaisseau contrôlé

vies	: Nombre initial de vies du vaisseau contrôlé
pvMax	: Nombre maximum de points de vie du vaisseau contrôlé
bombs	: Nombre initial de bombes du vaisseau contrôlé
shootingSpeed	: Vitesse de tir initiale du vaisseau contrôlé

5.10.3 Documentation des fonctions membres

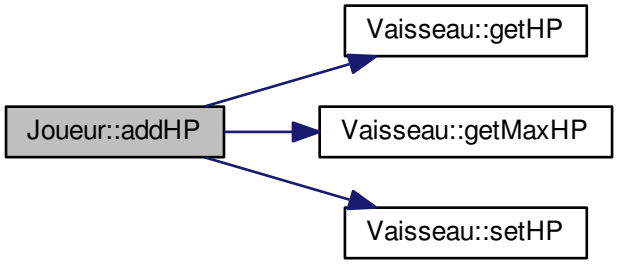
5.10.3.1 void Joueur : :addHP (int *hp*)

Ajouter des points de vie.

Paramètres

hp	: Points de vie à ajouter

Voici le graphe d'appel pour cette fonction :



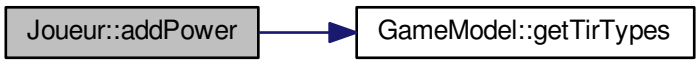
5.10.3.2 void Joueur : :addPower (int *power,* GameModel ∗ *model*)

Ajouter de la puissance.

Paramètres

power	: Puissance à ajouter
model	: Modéle du jeu

Voici le graphe d'appel pour cette fonction :



5.10.3.3 int Joueur : :getBombs ( ) const

Accesseur.

Accesseur de my_bombs

Renvoie

: La quantité de bombes actuelle du vaisseau contrôlé

5.10.3.4 int Joueur : :getNumberOfShoots ( ) const

Accesseur.

Accesseur de my_numberOfShoots

Renvoie

: Le nombre de tirs simultanés du vaisseau contrôlé (dépend de la puissance)

5.10.3.5 int Joueur : :getPower ( ) const

Accesseur.

Accesseur de my_power

Renvoie

: La puissance actuelle du vaisseau contrôlé

5.10.3.6 string Joueur : :getStringBombs ( ) const

Accesseur.

Accesseur de my_bombs

Renvoie

: La quantité de bombes actuelle du vaisseau contrôlé, en chaîne de caractères

5.10.3.7 string Joueur : :getStringPower ( ) const

Accesseur.

Accesseur de my_power

Renvoie

: La puissance actuelle du vaisseau contrôlé, en chaîne de caractères

5.10.3.8 int Joueur : :getVies ( ) const

Accesseur.

Accesseur de my_vies

Renvoie

: Le nombre de vies actuel du vaisseau contrôlé

5.10.3.9 bool Joueur : :isHitByEnemy (GameModel * *model*)

Indiquer si le vaisseau contrôlé est touché

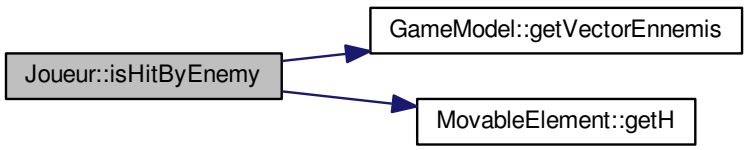
Paramètres

model	: Modéle du jeu

Renvoie

: L'état du vaisseau contrôlé (**true** si le vaisseau contrôlé est touché, **false** sinon)

Voici le graphe d'appel pour cette fonction :



5.10.3.10 vector< Tir * > : :iterator Joueur : :isShot (GameModel * *model*) const [virtual]

Indiquer le tir touchant le vaisseau.

Paramètres

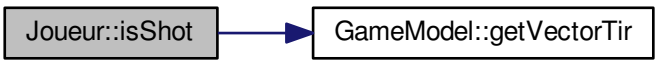
model	: Modéle du jeu

Renvoie

: Le tir touchant le vaisseau contrôlé

Implémente Vaisseau.

Voici le graphe d'appel pour cette fonction :



5.10.3.11 void Joueur : :shoot (GameModel * *model*) [virtual]

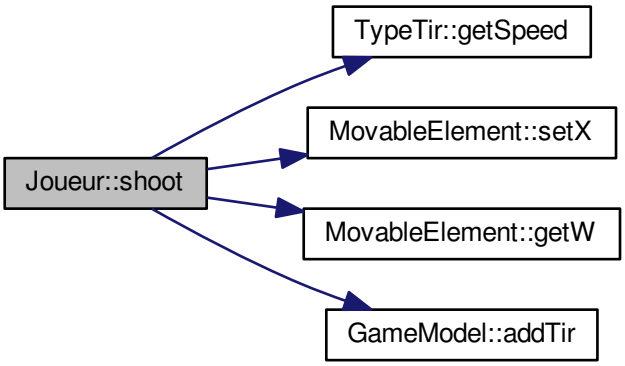
Faire tirer le vaisseau contrôlé

Paramètres

model	: Modéle du jeu

Implémente Vaisseau.

Voici le graphe d'appel pour cette fonction :



5.10.3.12 void Joueur : :useBomb (GameModel * *model*)

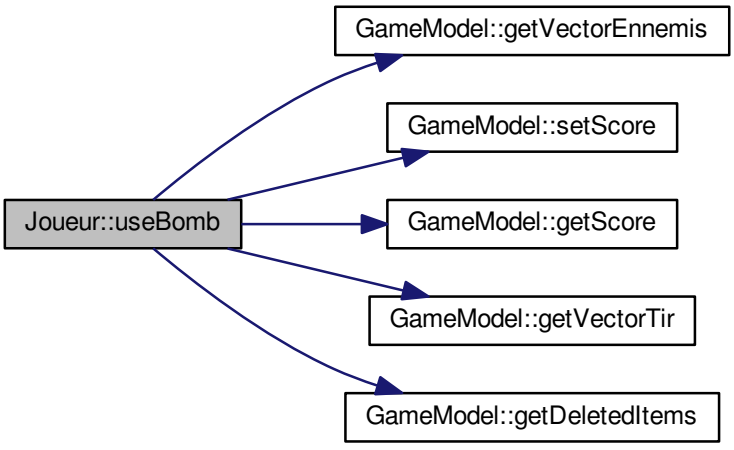
Utiliser une bombe.

Méthode permettant de détruire tous les ennemis à l'écran

Paramètres

model	: Modéle du jeu

Voici le graphe d'appel pour cette fonction :



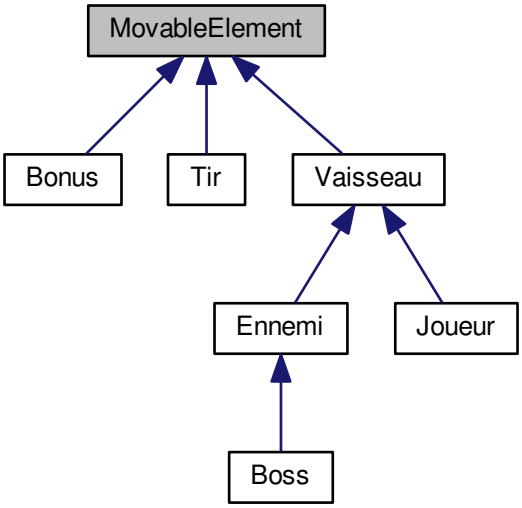
La documentation de cette classe a été générée à partir des fichiers suivants :
– Joueur.h
– Joueur.cc

5.11 Référence de la classe MovableElement

Classe représentant les éléments mobiles.

`#include <MovableElement.h>`

Graphe d'héritage de MovableElement :



Fonctions membres publiques

– MovableElement ()
  Constructeur par défaut.
– MovableElement (int x, int y, int dx, int dy, int w, int h)
  Constructeur paramétré
– virtual ~MovableElement ()
  Destructeur.
– virtual void moveLeft ()
  Déplacer l'élément mobile vers la gauche.
– virtual void moveRight ()
  Déplacer l'élément mobile vers la droite.
– virtual void moveDown ()
  Déplacer l'élément mobile vers le bas.
– virtual void moveUp ()
  Déplacer l'élément mobile vers le haut.
– virtual void move ()
  Déplacer l'élément mobile.
– int getX () const
  Accesseur.
– int getY () const
  Accesseur.
– int getDY () const

Accesseur.
- int getW () const
  Accesseur.
- int getH () const
  Accesseur.
- void setX (int x)
  Définir l'abscisse de l'élément mobile.
- void setY (int y)
  Définir l'ordonnée de l'élément mobile.

Attributs protégés

- int my_x
- int my_y
- int my_dx
- int my_dy
- int my_w
- int my_h

5.11.1 Description détaillée

Classe représentant les éléments mobiles.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.11.2 Documentation des constructeurs et destructeur

5.11.2.1 MovableElement : :MovableElement (int *x*, int *y*, int *dx*, int *dy*, int *w*, int *h*)

Constructeur paramétré

Paramètres

x	: Abscisse de l'emplacement de l'élément mobile dans le Modéle
y	: Ordonnée de l'emplacement de l'élément mobile dans le Modéle
dx	: Vitesse (abscisse) de l'élément mobile
dy	: Vitesse (ordonnée) de l'élément mobile
w	: Largeur de l'élément mobile
h	: Hauteur de l'élément mobile

5.11.3 Documentation des fonctions membres

5.11.3.1 int MovableElement : :getDY ( ) const

Accesseur.

Accesseur de my_dy

Renvoie

: La vitesse (ordonnée) de l'élément mobile

5.11.3.2 int MovableElement : :getH ( ) const

Accesseur.

Accesseur de my_h

Renvoie

: La hauteur de l'élément mobile de l'élément mobile

5.11.3.3 int MovableElement : :getW ( ) const

Accesseur.

Accesseur de my_w

Renvoie

: La largeur de l'élément mobile de l'élément mobile

5.11.3.4 int MovableElement : :getX ( ) const

Accesseur.

Accesseur de my_x

Renvoie

: L'abscisse de l'élément mobile

5.11.3.5 int MovableElement : :getY ( ) const

Accesseur.

Accesseur de my_y

Renvoie

: L'ordonnée de l'élément mobile

5.11.3.6 void MovableElement : :move ( ) [virtual]

Déplacer l'élément mobile.

Méthode permettant de déplacer l'élément mobile en abscisse et en ordonnée

Réimplémentée dans Ennemi, et Boss.

5.11.4 Documentation des données membres

5.11.4.1 int MovableElement : :my_dx [protected]

Vitesse (abscisse) de l'élément mobile

5.11.4.2 int MovableElement : :my_dy [protected]

Vitesse (ordonnée) de l'élément mobile

5.11.4.3 int MovableElement : :my_h [protected]

Hauteur de l'élément mobile

5.11.4.4 int MovableElement : :my_w [protected]

Largeur de l'élément mobile

5.11.4.5 int MovableElement : :my_x [protected]

Abscisse de l'élément mobile

5.11.4.6 int MovableElement : :my_y [protected]

Ordonnée de l'élément mobile

La documentation de cette classe a été générée à partir des fichiers suivants :
– MovableElement.h
– MovableElement.cc

5.12 Référence de la classe Niveau

Classe représentant les niveaux du jeu.

`#include <Niveau.h>`

Fonctions membres publiques

– Niveau ()
  Constructeur par défaut.
– void initNiveau (GameModel ∗model)
  Initialise le niveau.
– const std : :multimap< float, TypeEnnemi ∗ > ∗ getMap () const
  Accesseur.
– int getID () const
  Accesseur.

5.12.1 Description détaillée

Classe représentant les niveaux du jeu.

Auteur

Quentin HARSCOËT

5.12.2 Documentation des fonctions membres

5.12.2.1 int Niveau : :getID ( ) const

Accesseur.

Accesseur de idNiveau

Renvoie

: L'identifiant du niveau

5.12.2.2 const multimap< float, TypeEnnemi * > * Niveau : :getMap ( ) const

Accesseur.

Accesseur de ennemis

Renvoie

: L'adresse de la multimap contenant les instant où les ennemis appraîssent, ainsi que les types de ces derniers

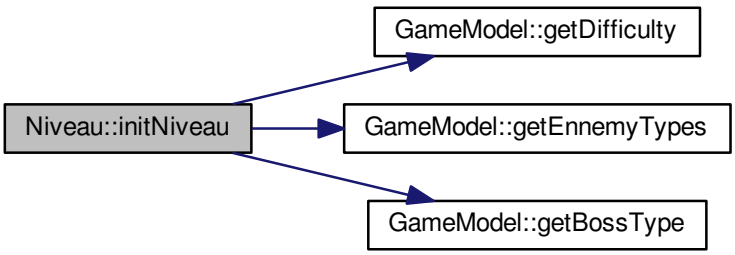
5.12.2.3 void Niveau : :initNiveau (GameModel * *model*)

Initialise le niveau.

Paramètres

model	: Modèle du jeu

Voici le graphe d'appel pour cette fonction :



La documentation de cette classe a été générée à partir des fichiers suivants :
– Niveau.h
– Niveau.cc

5.13 Référence de la structure ShootPattern

Catégories de patterns.

`#include <TypeEnnemi.h>`

5.13.1 Description détaillée

Catégories de patterns.

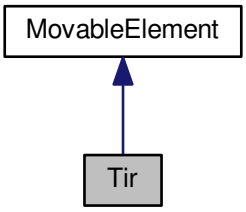
La documentation de cette structure a été générée à partir du fichier suivant :
– TypeEnnemi.h

5.14 Référence de la classe Tir

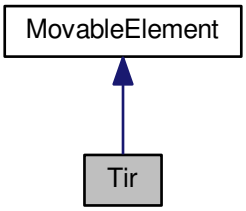
Classe représentant les tirs des Vaisseau.

`#include <Tir.h>`

Graphe d'héritage de Tir :



Graphe de collaboration de Tir :



Fonctions membres publiques

- Tir (int x, int y, bool fromPlayer, TypeTir ∗type)
  Constructeur paramétré
- Tir (int x, int y, int dx, int dy, bool fromPlayer, TypeTir ∗type)
  Constructeur paramétré
- virtual ∼Tir ()
  Destructeur.
- bool getPlayer () const
  Accesseur.
- TypeTir ∗ getType () const
  Accesseur.

Additional Inherited Members

5.14.1 Description détaillée

Classe représentant les tirs des Vaisseau.

Auteur

Quentin HARSCOËT

5.14.2 Documentation des constructeurs et destructeur

5.14.2.1 Tir : :Tir (int *x,* int *y,* bool *fromPlayer,* TypeTir ∗ *type*)

Constructeur paramétré

Paramètres

x	: Abscisse de l'emplacement du tir dans le Modéle
y	: Ordonnée de l'emplacement du tir dans le Modéle
fromPlayer	: Origine du tir (**true** si le tir vient du joueur, **false** sinon)
type	: Type du tir

5.14.2.2 Tir : :Tir (int *x,* int *y,* int *dx,* int *dy,* bool *fromPlayer,* TypeTir ∗ *type*)

Constructeur paramétré

Paramètres

x	: Abscisse de l'emplacement du tir dans le Modéle
y	: Ordonnée de l'emplacement du tir dans le Modéle
dx	: Vitesse (abscisse) du tir
dy	: Vitesse (ordonnée) du tir
fromPlayer	: Origine du tir (**true** si le tir vient du joueur, **false** sinon)
type	: Type du tir

5.14.3 Documentation des fonctions membres

5.14.3.1 bool Tir : :getPlayer ( ) const

Accesseur.

Accesseur de my_fromPlayer

Renvoie

: L'origine du tir (**true** si le tir vient du joueur, **false** sinon)

5.14.3.2 TypeTir ∗ Tir : :getType ( ) const

Accesseur.

Accesseur de my_type

Renvoie

: Le type du tir

La documentation de cette classe a été générée à partir des fichiers suivants :
– Tir.h
– Tir.cc

5.15 Référence de la classe TypeBonus

Classe représentant les différents types de Bonus.

`#include <TypeBonus.h>`

Fonctions membres publiques

- TypeBonus (BonusEffect effect, int value, float frequency, float scale)
  Constructeur paramétré
- BonusEffect getEffect () const
  Accesseur.
- int getValue () const
  Accesseur.
- float getFrequency () const
  Accesseur.
- float getScale () const
  Accesseur.

5.15.1 Description détaillée

Classe représentant les différents types de Bonus.

Auteur

Quentin HARSCOËT

5.15.2 Documentation des constructeurs et destructeur

5.15.2.1 TypeBonus : :TypeBonus (BonusEffect *effect,* int *value,* float *frequency,* float *scale*)

Constructeur paramétré

Paramètres

effect	: Effet/Catégorie du Bonus
value	: Valeur du Bonus (le gain apporté)
frequency	: Fréquence/probabilité d'apparition du bonus, comprise entre 0 et 1
scale	: Taille du Bonus

5.15.3 Documentation des fonctions membres

5.15.3.1 BonusEffect TypeBonus : :getEffect ( ) const

Accesseur.

Accesseur de my_effect

Renvoie

: L'effet/catégorie du Bonus

5.15.3.2 float TypeBonus : :getFrequency ( ) const

Accesseur.

Accesseur de my_frequency

Renvoie

: La fréquence/probabilité d'apparition du Bonus

5.15.3.3 float TypeBonus : :getScale ( ) const

Accesseur.

Accesseur de my_scale

Renvoie

: La taille du Bonus

5.15.3.4 int TypeBonus : :getValue ( ) const

Accesseur.

Accesseur de my_value

Renvoie

: La valeur du Bonus (le gain apporté)

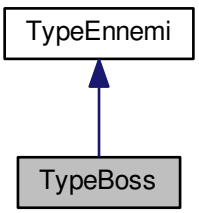
La documentation de cette classe a été générée à partir des fichiers suivants :
– TypeBonus.h
– TypeBonus.cc

5.16 Référence de la classe TypeBoss

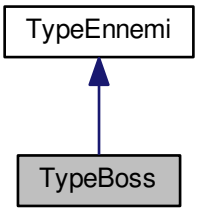
Classe représentant les différents types de Boss.

`#include <TypeBoss.h>`

Graphe d'héritage de TypeBoss :



Graphe de collaboration de TypeBoss :



Fonctions membres publiques

– TypeBoss (int score, int speed, float shootingSpeed, int hp, int w, int h, TypeTir ∗typeTir, ShootPattern shoot-Pattern, std : :string filename, float changePatternDelay)
Constructeur paramétré
– float getChangePatternDelay () const
Accesseur.

5.16.1 Description détaillée

Classe représentant les différents types de Boss.

Auteur

Quentin HARSCOËT

5.16.2 Documentation des constructeurs et destructeur

5.16.2.1 TypeBoss : :TypeBoss (int *score,* int *speed,* float *shootingSpeed,* int *hp,* int *w,* int *h,* TypeTir ∗ *typeTir,* ShootPattern *shootPattern,* std : :string *filename,* float *changePatternDelay*)

Constructeur paramétré

Paramètres

score	: Points rapportés par le Boss
speed	: Vitesse du Boss
shootingSpeed	: Vitesse de tir du Boss
hp	: Points de vie du Boss
w	: Largeur du Boss
h	: Hauteur du Boss
typeTir	: Type de tir du Boss
shootPattern	: Pattern de tir du Boss
filename	: Nom du fichier du sprite du Boss
changePattern-Delay	: Intervalle de temps entre deux changements de pattern du Boss

5.16.3 Documentation des fonctions membres

5.16.3.1 float TypeBoss : :getChangePatternDelay ( ) const

Accesseur.

Accesseur de my_changePatternDelay

Renvoie

: L'intervalle de temps entre deux changements de pattern du Boss

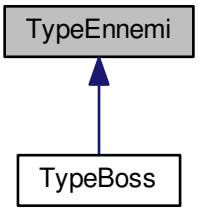
La documentation de cette classe a été générée à partir des fichiers suivants :
– TypeBoss.h
– TypeBoss.cc

5.17 Référence de la classe TypeEnnemi

Classe représentant les différents types d'Ennemi.

`#include <TypeEnnemi.h>`

Graphe d'héritage de TypeEnnemi :



Fonctions membres publiques

– TypeEnnemi ()
  Constructeur par défaut.
– TypeEnnemi (int score, int speed, float shootingSpeed, int hp, int w, int h, TypeTir ∗typeTir, ShootPattern shoot-Pattern, std : :string filename)
  Constructeur paramétré
– virtual ∼TypeEnnemi ()
  Destructeur.
– int getScore () const
  Accesseur.
– int getSpeed () const
  Accesseur.
– float getShootingSpeed () const
  Accesseur.
– int getHP () const
  Accesseur.
– int getW () const
  Accesseur.
– int getH () const
  Accesseur.
– std : :string getFilename () const
  Accesseur.
– TypeTir ∗ getTypeTir () const
  Accesseur.

– ShootPattern getShootPattern () const
Accesseur.

5.17.1 Description détaillée

Classe représentant les différents types d'Ennemi.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.17.2 Documentation des constructeurs et destructeur

5.17.2.1 TypeEnnemi : :TypeEnnemi (int *score,* int *speed,* float *shootingSpeed,* int *hp,* int *w,* int *h,* TypeTir * *typeTir,* ShootPattern *shootPattern,* std : :string *filename*)

Constructeur paramétré

Paramètres

score	: Points rapportés par l'Ennemi
speed	: Vitesse de l'Ennemi
shootingSpeed	: Vitesse de tir de l'Ennemi
hp	: Points de vie de l'Ennemi
w	: Largeur de l'Ennemi
h	: Hauteur de l'Ennemi
typeTir	: Type de tir de l'Ennemi
shootPattern	: Pattern de tir de l'Ennemi
filename	: Nom du fichier du sprite de l'Ennemi

5.17.3 Documentation des fonctions membres

5.17.3.1 string TypeEnnemi : :getFilename ( ) const

Accesseur.

Accesseur de my_filename

Renvoie

: La nom du fichier du sprite de l'Ennemi

5.17.3.2 int TypeEnnemi : :getH ( ) const

Accesseur.

Accesseur de my_h

Renvoie

: La hauteur de l'Ennemi

5.17.3.3 int TypeEnnemi : :getHP ( ) const

Accesseur.

Accesseur de my_hp

Renvoie

: Les points de vie de l'Ennemi

5.17.3.4 int TypeEnnemi : :getScore ( ) const

Accesseur.

Accesseur de my_score

Renvoie

: Les points rapportés par l'Ennemi

5.17.3.5 float TypeEnnemi : :getShootingSpeed ( ) const

Accesseur.

Accesseur de my_shootingSpeed

Renvoie

: La vitesse de tir de l'Ennemi

5.17.3.6 ShootPattern TypeEnnemi : :getShootPattern ( ) const

Accesseur.

Accesseur de my_shootPattern

Renvoie

: Le pattern de tir de l'Ennemi

5.17.3.7 int TypeEnnemi : :getSpeed ( ) const

Accesseur.

Accesseur de my_speed

Renvoie

: La vitesse de l'Ennemi

5.17.3.8 TypeTir * TypeEnnemi : :getTypeTir ( ) const

Accesseur.

Accesseur de my_typeTir

Renvoie

: Le type de tir de l'Ennemi

5.17.3.9 int TypeEnnemi : :getW ( ) const

Accesseur.

Accesseur de my_w

Renvoie

: La largeur de l'Ennemi

La documentation de cette classe a été générée à partir des fichiers suivants :
– TypeEnnemi.h
– TypeEnnemi.cc

5.18 Référence de la classe TypeTir

Classe représentant les différents types de Tir.

`#include <TypeTir.h>`

Fonctions membres publiques

– TypeTir (int w, int h, int damages, int speed, std : :string filename)
Constructeur paramétré
– std : :string getFilename () const
Accesseur.
– int getH () const
Accesseur.
– int getW () const
Accesseur.
– int getDamages () const
Accesseur.
– int getSpeed () const
Accesseur.

5.18.1 Description détaillée

Classe représentant les différents types de Tir.

Auteur

Quentin HARSCOËT

5.18.2 Documentation des constructeurs et destructeur

5.18.2.1 TypeTir : :TypeTir (int *w,* int *h,* int *damages,* int *speed,* std : :string *filename*)

Constructeur paramétré

Paramètres

w	: Largeur du Tir
h	: Hauteur du Tir
damages	: Dommages infligés par le Tir
speed	: Vitesse du Tir

filename	: Nom du fichier du sprite du Tir

5.18.3 Documentation des fonctions membres

5.18.3.1 int TypeTir : :getDamages ( ) const

Accesseur.

Accesseur de my_w

Renvoie

: Les dommages infligés par le Tir

5.18.3.2 string TypeTir : :getFilename ( ) const

Accesseur.

Accesseur de my_filename

Renvoie

: Le nom du fichier du sprite du Tir

5.18.3.3 int TypeTir : :getH ( ) const

Accesseur.

Accesseur de my_h

Renvoie

: La hauteur du Tir

5.18.3.4 int TypeTir : :getSpeed ( ) const

Accesseur.

Accesseur de my_w

Renvoie

: La vitesse du Tir

5.18.3.5 int TypeTir : :getW ( ) const

Accesseur.

Accesseur de my_w

Renvoie

: La largeur du Tir

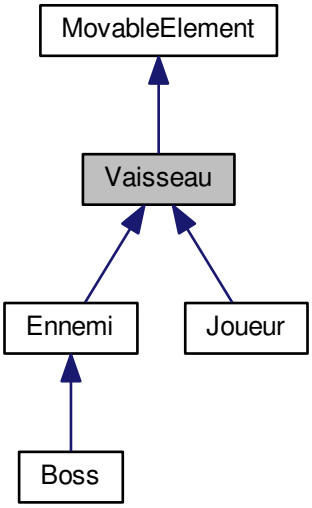
La documentation de cette classe a été générée à partir des fichiers suivants :
– TypeTir.h
– TypeTir.cc

5.19 Référence de la classe Vaisseau

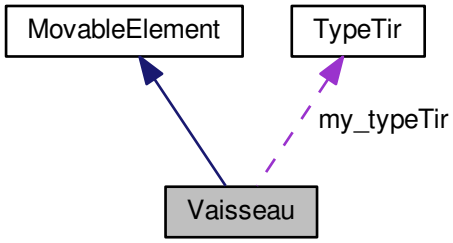
Classe représentant les vaisseaux.

`#include <Vaisseau.h>`

Graphe d'héritage de Vaisseau :



Graphe de collaboration de Vaisseau :



Fonctions membres publiques

- Vaisseau (int hp, TypeTir *type, float shootingSpeed)
  Constructeur paramétré
- virtual ~Vaisseau ()
  Destructeur.
- int getHP () const
  Accesseur.

- int getMaxHP () const
  Accesseur.
- TypeTir * getTypeTir () const
  Accesseur.
- bool getAlive () const
  Accesseur.
- float getShootingSpeed () const
  Accesseur.
- sf : :Clock * getClock ()
  Accesseur.
- void setHP (int i)
  Définir les points de vie du Vaisseau.
- void setTypeTir (TypeTir *type)
  Définir le type de tir du Vaisseau.
- void setAlive (bool b)
  Définir l'état du Vaisseau.
- virtual void shoot (GameModel *model)=0
  Faire tirer le Vaisseau (méthode virtuelle pure)
- virtual std : :vector< Tir * >
  : :iterator isShot (GameModel *model) const =0
  Indiquer le tir touchant le Vaisseau.

Attributs protégés

- TypeTir * my_typeTir
- sf : :Clock my_clock
- float my_shootingSpeed

5.19.1 Description détaillée

Classe représentant les vaisseaux.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

5.19.2 Documentation des constructeurs et destructeur

5.19.2.1 Vaisseau : :Vaisseau (int *hp*, TypeTir * *type*, float *shootingSpeed*)

Constructeur paramétré

Paramètres

hp	: Points de vie initiaux du Vaisseau
type	: Type de tir initial du Vaisseau
shootingSpeed	: Cadence de tir initiale du Vaisseau

5.19.3 Documentation des fonctions membres

5.19.3.1 bool Vaisseau : :getAlive ( ) const

Accesseur.

Accesseur de my_isAlive

Renvoie

: L'état actuel du Vaisseau (**true** si le Vaisseau n'est pas détruit, **false** sinon)

5.19.3.2 sf : :Clock * Vaisseau : :getClock ( )

Accesseur.

Accesseur de my_clock

Renvoie

: La Clock interne au Vaisseau

5.19.3.3 int Vaisseau : :getHP ( ) const

Accesseur.

Accesseur de my_hp

Renvoie

: Les points de vie actuels du Vaisseau

5.19.3.4 int Vaisseau : :getMaxHP ( ) const

Accesseur.

Accesseur de my_pvMax

Renvoie

: Les points de vie maximum du Vaisseau

5.19.3.5 float Vaisseau : :getShootingSpeed ( ) const

Accesseur.

Accesseur de my_shootingSpeed

Renvoie

: La cadence de tir actuelle du Vaisseau

5.19.3.6 TypeTir * Vaisseau : :getTypeTir ( ) const

Accesseur.

Accesseur de my_typeTir

Renvoie

: Le type de tir actuel du Vaisseau

5.19.3.7 virtual std : :vector<Tir*> : :iterator Vaisseau : :isShot (GameModel * *model*) const [pure virtual]

Indiquer le tir touchant le Vaisseau.

Paramètres

model	: Modéle du jeu (méthode virtuelle pure)

Renvoie

: Le tir touchant le vaisseau contrôlé

Implémenté dans Ennemi, et Joueur.

5.19.3.8 void Vaisseau : :setAlive (bool *b*)

Définir l'état du Vaisseau.

Paramètres

b	: Nouvel état du Vaisseau (**true** si le Vaisseau n'est pas détruit, **false** sinon)

5.19.3.9 void Vaisseau : :setHP (int *i*)

Définir les points de vie du Vaisseau.

Paramètres

i	: Nouveaux points de vie du Vaisseau

5.19.3.10 void Vaisseau : :setTypeTir (TypeTir * *type*)

Définir le type de tir du Vaisseau.

Paramètres

type	: Nouveau type de tir du Vaisseau

5.19.3.11 virtual void Vaisseau : :shoot (GameModel * *model*) [pure virtual]

Faire tirer le Vaisseau (méthode virtuelle pure)

Paramètres

model	: Modèle du jeu

Implémenté dans Ennemi, et Joueur.

5.19.4 Documentation des données membres

5.19.4.1 sf : :Clock Vaisseau : :my_clock [protected]

Pendule interne au Vaisseau, pour y gérer le temps

5.19.4.2 float Vaisseau : :my_shootingSpeed [protected]

Cadence de tir, en secondes (un tir toutes les x secondes)

5.19.4.3 TypeTir* Vaisseau : :my_typeTir [protected]

Pointeur vers le type des tirs du Vaisseau

La documentation de cette classe a été générée à partir des fichiers suivants :
– Vaisseau.h
– Vaisseau.cc

Chapitre 6

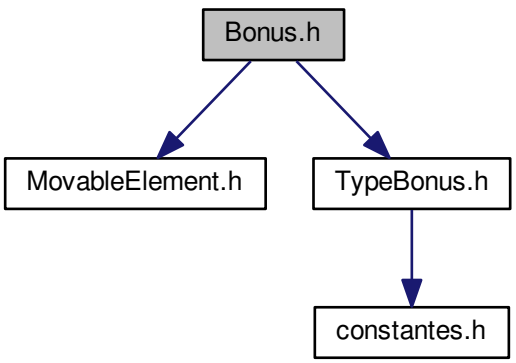
Documentation des fichiers

6.1 Référence du fichier Bonus.h

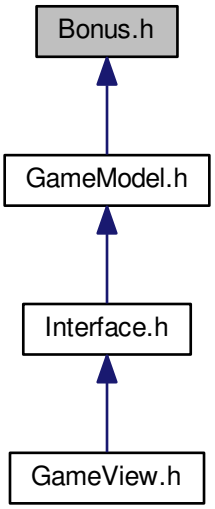
Déclaration de la class Bonus.

```
#include "MovableElement.h"
#include "TypeBonus.h"
```

Graphe des dépendances par inclusion de Bonus.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Bonus
Classe représentant les bonii à ramasser.

6.1.1 Description détaillée

Déclaration de la class Bonus.

Auteur

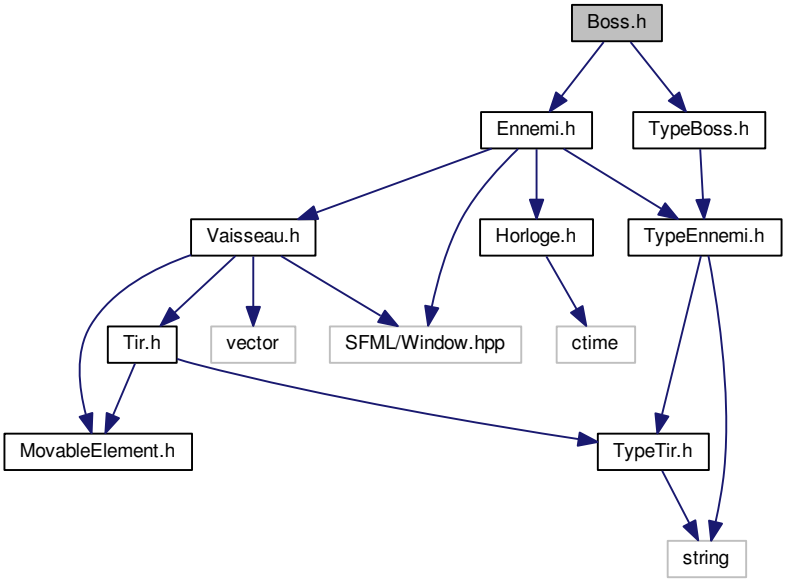
Théo CHASSAIGNE
Quentin HARSCOËT

6.2 Référence du fichier Boss.h

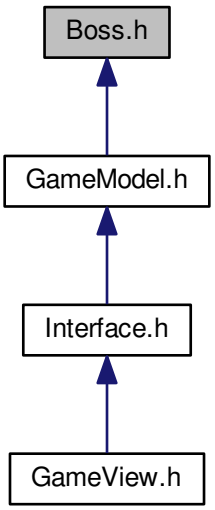
Déclaration de la class Boss.

```
#include "Ennemi.h"
#include "TypeBoss.h"
```

Graphe des dépendances par inclusion de Boss.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Boss
Classe représentant les boss de fin de niveau.

6.2.1 Description détaillée

Déclaration de la class Boss.

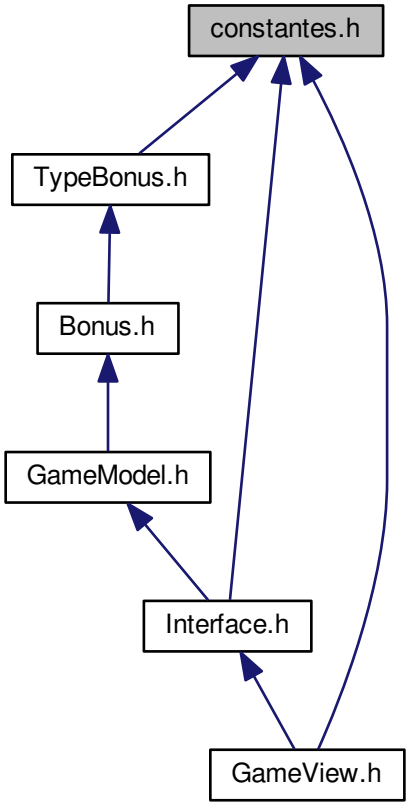
Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

6.3 Référence du fichier constantes.h

Déclaration des constantes utilisées.

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

- #define MODEL_WIDTH 768
- #define MODEL_HEIGHT 768
- #define VIEW_WIDTH 768
- #define VIEW_HEIGHT 768
- #define SHEEP_WIDTH 200
- #define SHEEP_HEIGHT 309
- #define SHEEP_SCALE 0.2
- #define ENEMY_SCALE 0.3
- #define SHEEP_SPEED 10

- #define MOVABLE_ELEMENT_SPEED 10
- #define NB_ENNEMY_TYPES 3
- #define NB_TIRS_TYPES 7
- #define NB_BONUS_TYPES 5
- #define NB_STRINGS 26
- #define VIES 0
- #define PV 1
- #define NIVEAU 2
- #define BOMBES 3
- #define PUISSANCE 4
- #define JOUER 5
- #define HIGHSCORES 6
- #define SETTINGS 7
- #define QUIT 8
- #define PAUSE 9
- #define RESUME 10
- #define BACKTOMENU 11
- #define NOSCORE 12
- #define ENTERNAME 13
- #define FINISHLEVEL 14
- #define GAMEOVER 15
- #define DEAD 16
- #define SOUNDLEVEL 17
- #define DIFFICULTY 18
- #define LIVESNUMBER 19
- #define LANGUAGE 20
- #define NORMAL 21
- #define HARD 22
- #define VERYHARD 23
- #define MYLANGUE 24
- #define SOUNDPACK 25

6.3.1 Description détaillée

Déclaration des constantes utilisées.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

6.3.2 Documentation des macros

6.3.2.1 #define BACKTOMENU 11

Index de la chaîne "Retourner au Menu"

6.3.2.2 #define BOMBES 3

Index de la chaîne "Bombes"

6.3.2.3 #define DEAD 16

Index de la chaîne "Vous avez perdu une vie ! "

6.3.2.4 #define DIFFICULTY 18

Index de la chaîne "Difficulté"

6.3.2.5 #define ENEMY_SCALE 0.3

Echelle des Ennemi

6.3.2.6 #define ENTERNAME 13

Index de la chaîne "Entrez votre nom"

6.3.2.7 #define FINISHLEVEL 14

Index de la chaîne "Vous avez fini un niveau !"

6.3.2.8 #define GAMEOVER 15

Index de la chaîne "Partie Terminée !"

6.3.2.9 #define HARD 22

Index de la chaîne "Difficile"

6.3.2.10 #define HIGHSCORES 6

Index de la chaîne "Meilleurs scores"

6.3.2.11 #define JOUER 5

Index de la chaîne "Jouer"

6.3.2.12 #define LANGUAGE 20

Index de la chaîne "Langue"

6.3.2.13 #define LIVESNUMBER 19

Index de la chaîne "Nombre de vies"

6.3.2.14 #define MODEL_HEIGHT 768

Hauteur du Modèle

6.3.2.15 #define MODEL_WIDTH 768

Largeur du Modèle

6.3.2.16 #define MOVABLE_ELEMENT_SPEED 10

Vitesse par défaut des MovableElement

6.3.2.17 #define MYLANGUE 24

Index de la chaîne indiquant la langue

6.3.2.18 #define NB_BONUS_TYPES 5

Nombre de TypeBonus

6.3.2.19 #define NB_ENNEMY_TYPES 3

Nombre de TypeEnnemi

6.3.2.20 #define NB_STRINGS 26

Nombre d'entrées du tableau contenant les textes du jeu

6.3.2.21 #define NB_TIRS_TYPES 7

Nombre de TypeTir

6.3.2.22 #define NIVEAU 2

Index de la chaîne "Niveau"

6.3.2.23 #define NORMAL 21

Index de la chaîne "Normal"

6.3.2.24 #define NOSCORE 12

Index de la chaîne "Pas de score"

6.3.2.25 #define PAUSE 9

Index de la chaîne "Pause"

6.3.2.26 #define PUISSANCE 4

Index de la chaîne "Puissance"

6.3.2.27 #define PV 1

Index de la chaîne "PV"

6.3.2.28 #define QUIT 8

Index de la chaîne "Quitter"

6.3.2.29 #define RESUME 10

Index de la chaîne "Continuer"

6.3.2.30 #define SETTINGS 7

Index de la chaîne "Paramètres"

6.3.2.31 #define SHEEP_HEIGHT 309

Hauteur du Joueur

6.3.2.32 #define SHEEP_SCALE 0.2

Echelle du Joueur

6.3.2.33 #define SHEEP_SPEED 10

Vitesse du Joueur

6.3.2.34 #define SHEEP_WIDTH 200

Largeur du Joueur

6.3.2.35 #define SOUNDLEVEL 17

Index de la chaîne "Volume sonore"

6.3.2.36 #define SOUNDPACK 25

Index de la chaîne "Pack de son"

6.3.2.37 #define VERYHARD 23

Index de la chaîne "Très difficile"

6.3.2.38 #define VIES 0

Index de la chaîne "Vies"

6.3.2.39 #define VIEW_HEIGHT 768

Hauteut de la Vue

6.3.2.40 #define VIEW_WIDTH 768

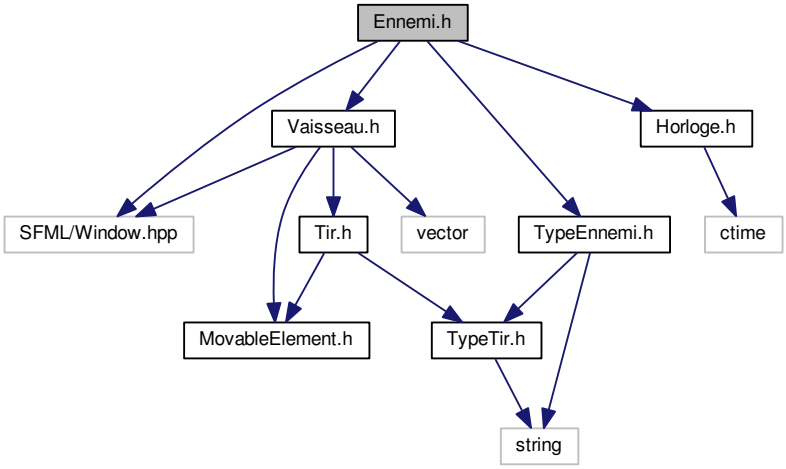
Largeur de la Vue

6.4 Référence du fichier Ennemi.h

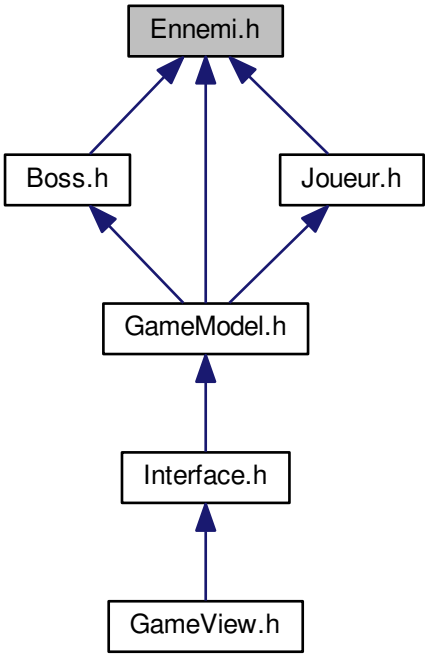
Déclaration de la class Ennemi.

```
#include <SFML/Window.hpp>
#include "Vaisseau.h"
#include "TypeEnnemi.h"
#include "Horloge.h"
```

Graphe des dépendances par inclusion de Ennemi.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Ennemi
Classe représentant les ennemis du joueur.

6.4.1 Description détaillée

Déclaration de la class Ennemi.

Auteur

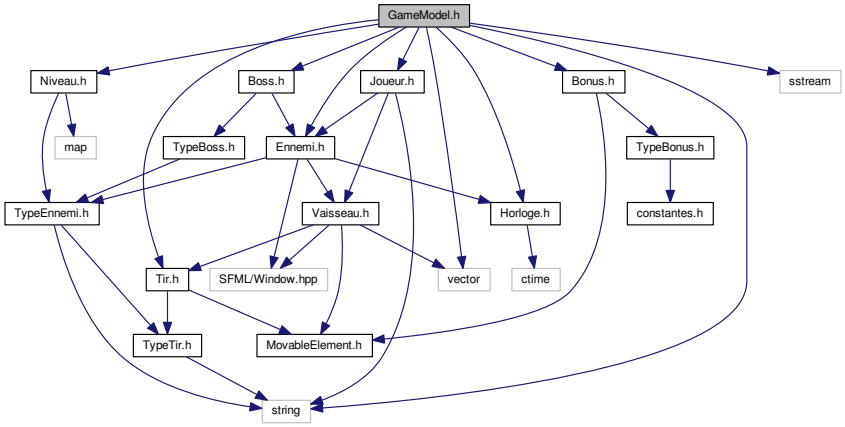
Théo CHASSAIGNE
Quentin HARSCOËT

6.5 Référence du fichier GameModel.h

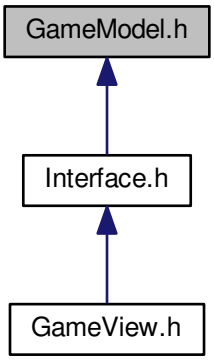
Déclaration de la classe GameModel.

```
#include "Tir.h"
#include "Ennemi.h"
#include "Boss.h"
#include "Joueur.h"
#include "Bonus.h"
#include "Niveau.h"
#include "Horloge.h"
#include <vector>
#include <string>
#include <sstream>
```

Graphe des dépendances par inclusion de GameModel.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class GameModel
Classe représentant le Modèle du jeu.

6.5.1 Description détaillée

Déclaration de la classe GameModel.

Auteur

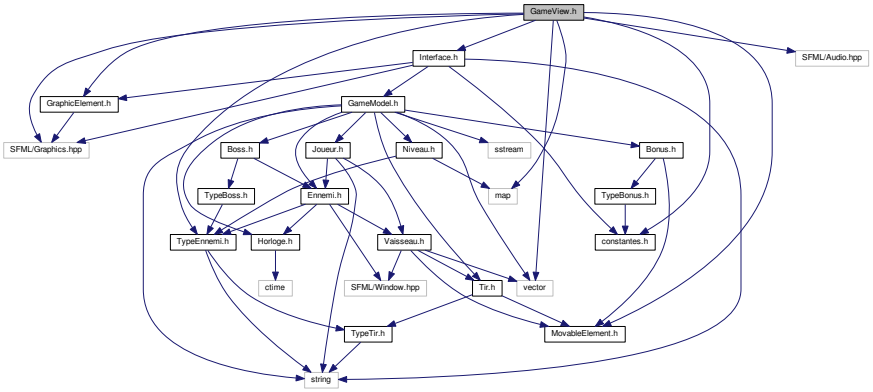
Théo CHASSAIGNE
Quentin HARSCOËT

6.6 Référence du fichier GameView.h

Déclaration de la classe GameView.

```
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <map>
#include <vector>
#include "GraphicElement.h"
#include "MovableElement.h"
#include "TypeEnnemi.h"
#include "constantes.h"
#include "Interface.h"
```

Graphe des dépendances par inclusion de GameView.h :



Classes

– class GameView
Classe représentant la Vue du jeu.

6.6.1 Description détaillée

Déclaration de la classe GameView.

Auteur

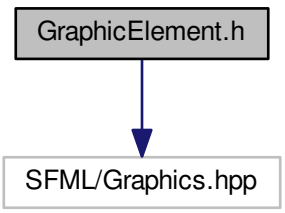
Théo CHASSAIGNE
Quentin HARSCOËT

6.7 Référence du fichier GraphicElement.h

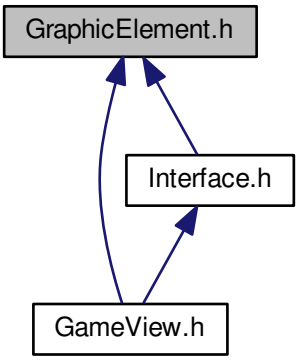
Déclaration de la classe GraphicElement.

```
#include <SFML/Graphics.hpp>
```

Graphe des dépendances par inclusion de GraphicElement.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class GraphicElement
Classe représentant les éléments à afficher.

6.7.1 Description détaillée

Déclaration de la classe GraphicElement.

Auteur

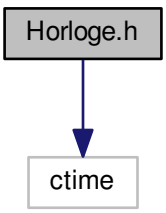
Théo CHASSAIGNE
Quentin HARSCOËT

6.8 Référence du fichier Horloge.h

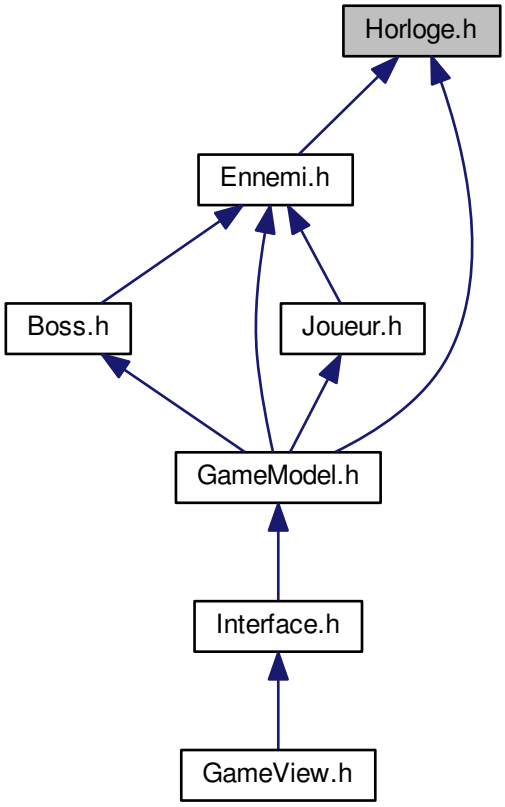
Déclaration de la classe Horloge.

```
#include <ctime>
```

Graphe des dépendances par inclusion de Horloge.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Horloge
Classe représentant le temps dans le jeu.

6.8.1 Description détaillée

Déclaration de la classe Horloge.

Auteur

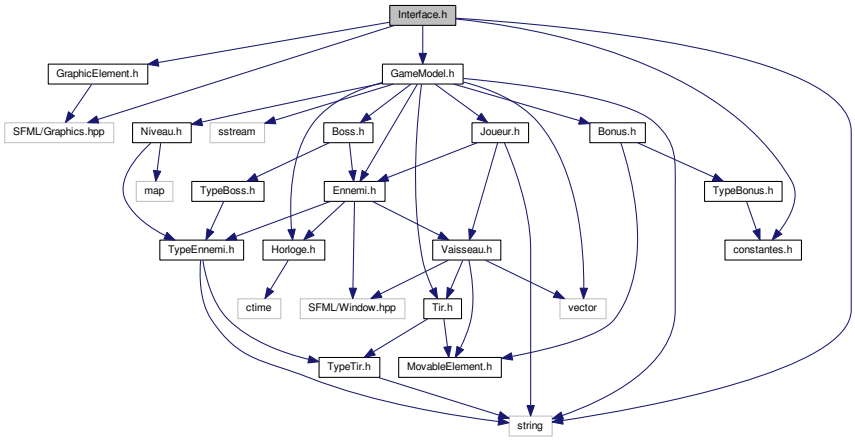
Théo CHASSAIGNE
Quentin HARSCOËT

6.9 Référence du fichier Interface.h

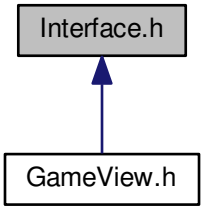
Déclaration de la classe Interface.

```
#include <SFML/Graphics.hpp>
#include <string>
#include "GameModel.h"
#include "GraphicElement.h"
#include "constantes.h"
```

Graphe des dépendances par inclusion de Interface.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Interface
Classe représentant le HUD (affichage tête haute)

6.9.1 Description détaillée

Déclaration de la classe Interface.

Auteur

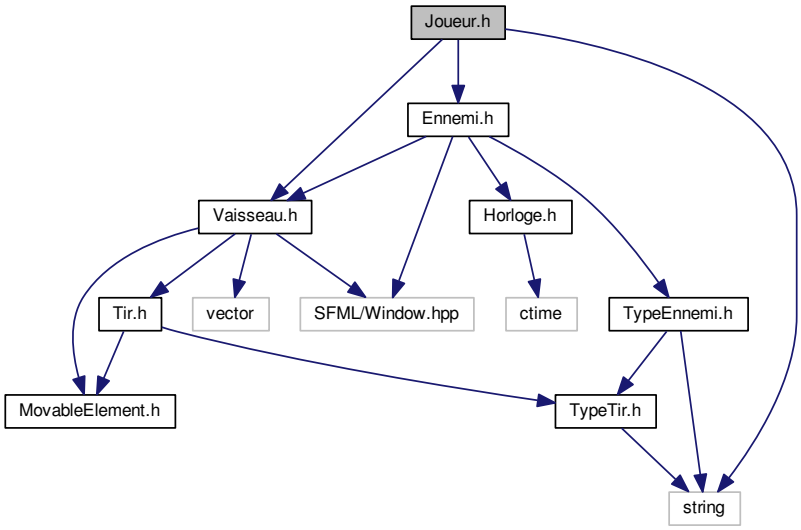
Quentin HARSCOËT

6.10 Référence du fichier Joueur.h

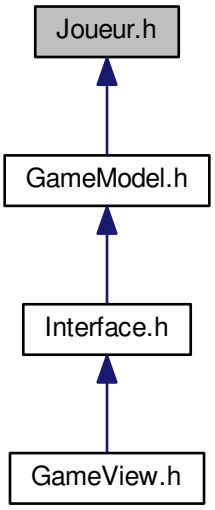
Déclaration de la classe Joueur.

```
#include "Vaisseau.h"
#include "Ennemi.h"
#include <string>
```

Graphe des dépendances par inclusion de Joueur.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Joueur
Classe représentant le vaisseau contrôlé

6.10.1 Description détaillée

Déclaration de la classe Joueur.

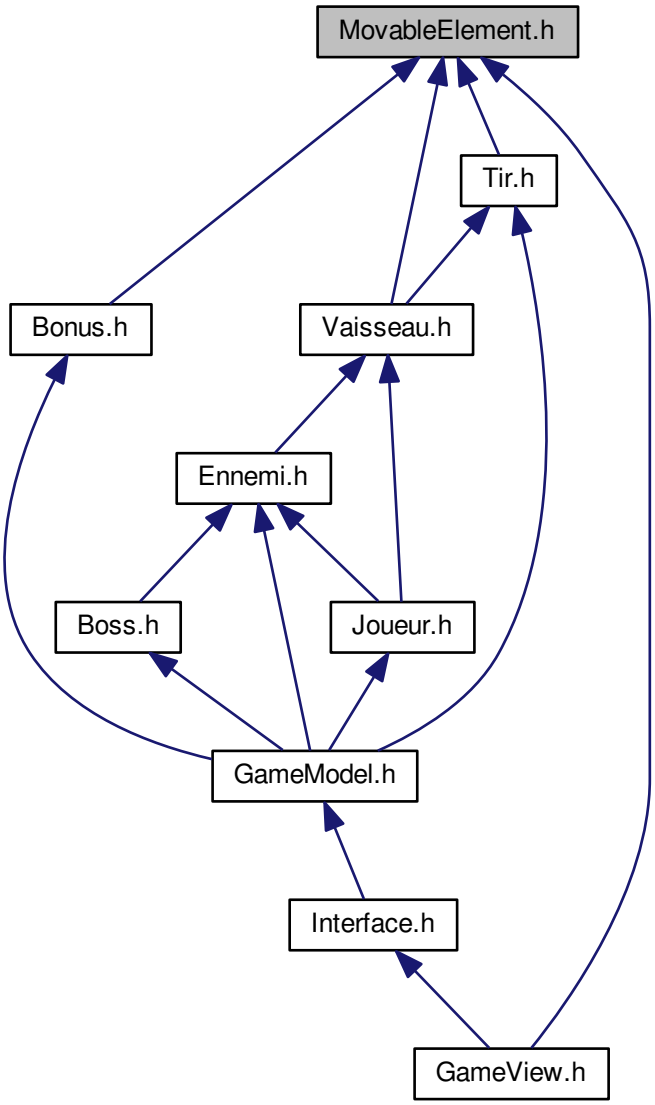
Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

6.11 Référence du fichier MovableElement.h

Déclaration de la classe MovableElement.

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class MovableElement
Classe représentant les éléments mobiles.

6.11.1 Description détaillée

Déclaration de la classe MovableElement.

Auteur

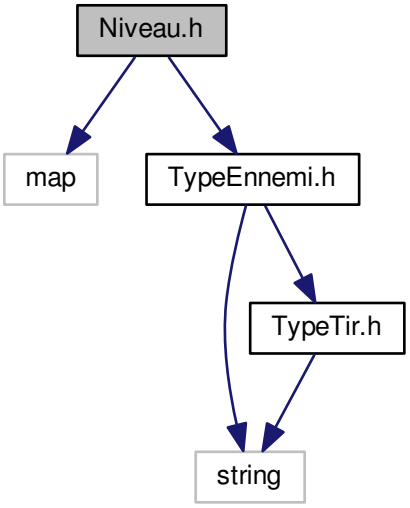
Théo CHASSAIGNE
Quentin HARSCOËT

6.12 Référence du fichier Niveau.h

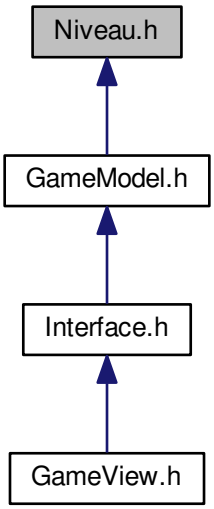
Déclaration de la classe Niveau.

```
#include <map>
#include "TypeEnnemi.h"
```

Graphe des dépendances par inclusion de Niveau.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Niveau
Classe représentant les niveaux du jeu.

6.12.1 Description détaillée

Déclaration de la classe Niveau.

Auteur

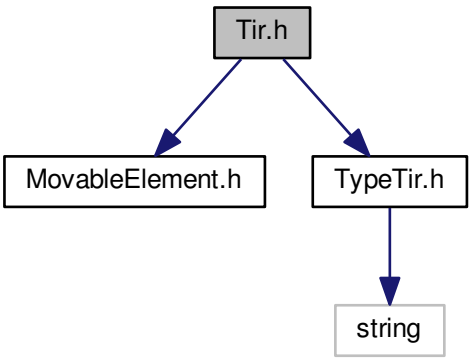
Quentin HARSCOËT

6.13 Référence du fichier Tir.h

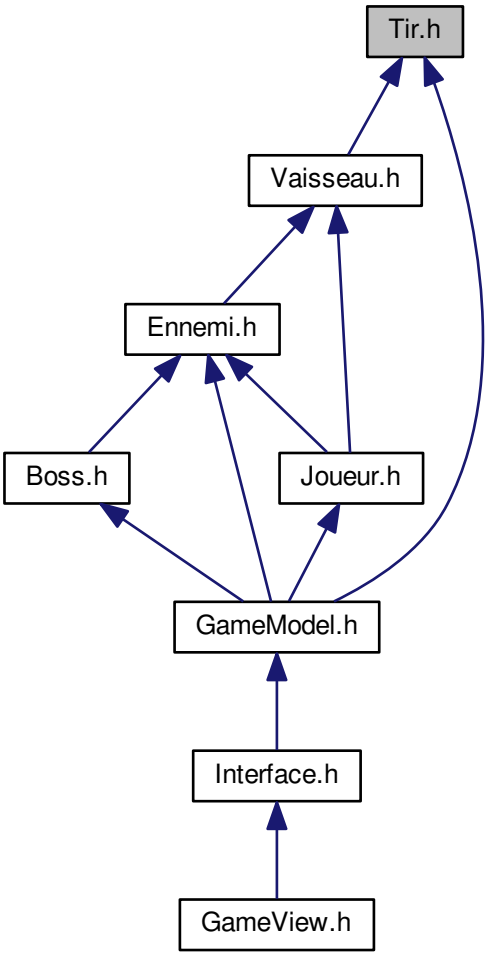
Déclaration de la classe Tir.

```
#include "MovableElement.h"
#include "TypeTir.h"
```

Graphe des dépendances par inclusion de Tir.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Tir
Classe représentant les tirs des Vaisseau.

6.13.1 Description détaillée

Déclaration de la classe Tir.

Auteur

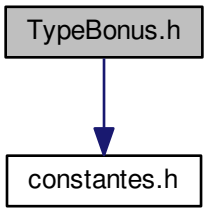
Quentin HARSCOËT

6.14 Référence du fichier TypeBonus.h

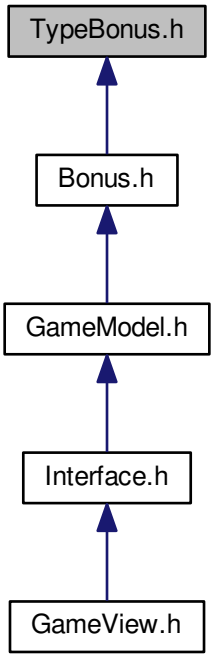
Déclaration de la classe TypeBonus et de l'énumération BonusEffect.

```
#include "constantes.h"
```

Graphe des dépendances par inclusion de TypeBonus.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class TypeBonus
Classe représentant les différents types de Bonus.

Énumérations

– enum BonusEffect { POWER, LIFE, BOMB, HP }

6.14.1 Description détaillée

Déclaration de la classe TypeBonus et de l'énumération BonusEffect.

Auteur

Quentin HARSCOËT

6.14.2 Documentation du type de l'énumération

6.14.2.1 enum BonusEffect

Valeurs énumérées

POWER Gain de puissance

LIFE Vie supplémentaire

BOMB Bombe supplémentaire

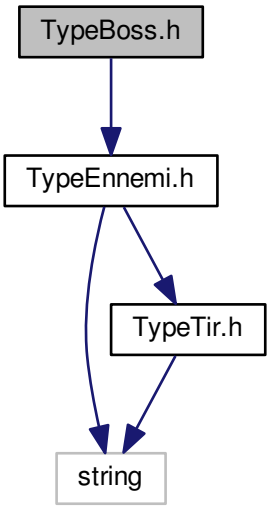
HP Gain de points de vie

6.15 Référence du fichier TypeBoss.h

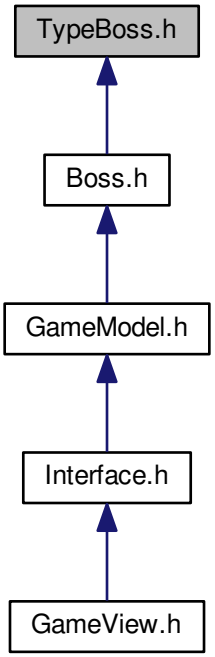
Déclaration de la classe TypeBoss.

```
#include "TypeEnnemi.h"
```

Graphe des dépendances par inclusion de TypeBoss.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class TypeBoss
Classe représentant les différents types de Boss.

6.15.1 Description détaillée

Déclaration de la classe TypeBoss.

Auteur

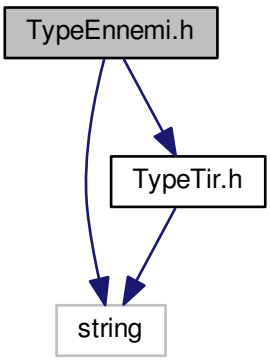
Quentin HARSCOËT

6.16 Référence du fichier TypeEnnemi.h

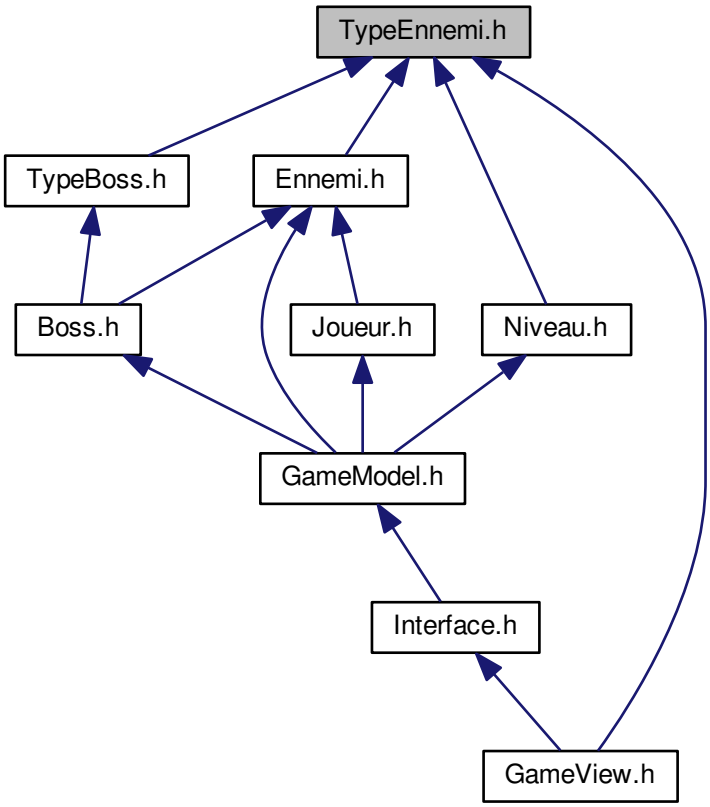
Déclaration de la classe TypeEnnemi et de l'énumération ShootPattern.

```
#include <string>
#include "TypeTir.h"
```

Graphe des dépendances par inclusion de TypeEnnemi.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class TypeEnnemi
Classe représentant les différents types d'Ennemi.

Énumérations

– enum ShootPattern { SIMPLE, M_TRIPLE, TOPLAYER, HALFCIRCLE }

6.16.1 Description détaillée

Déclaration de la classe TypeEnnemi et de l'énumération ShootPattern.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

6.16.2 Documentation du type de l'énumération

6.16.2.1 enum ShootPattern

Valeurs énumérées

SIMPLE Tir simple en ligne droite

M_TRIPLE Tir triple, "en triangle"

TOPLAYER Tir dirigé vers le Joueur

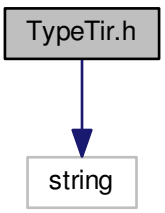
HALFCIRCLE Tir en demi-cercle, devant le Vaisseau

6.17 Référence du fichier TypeTir.h

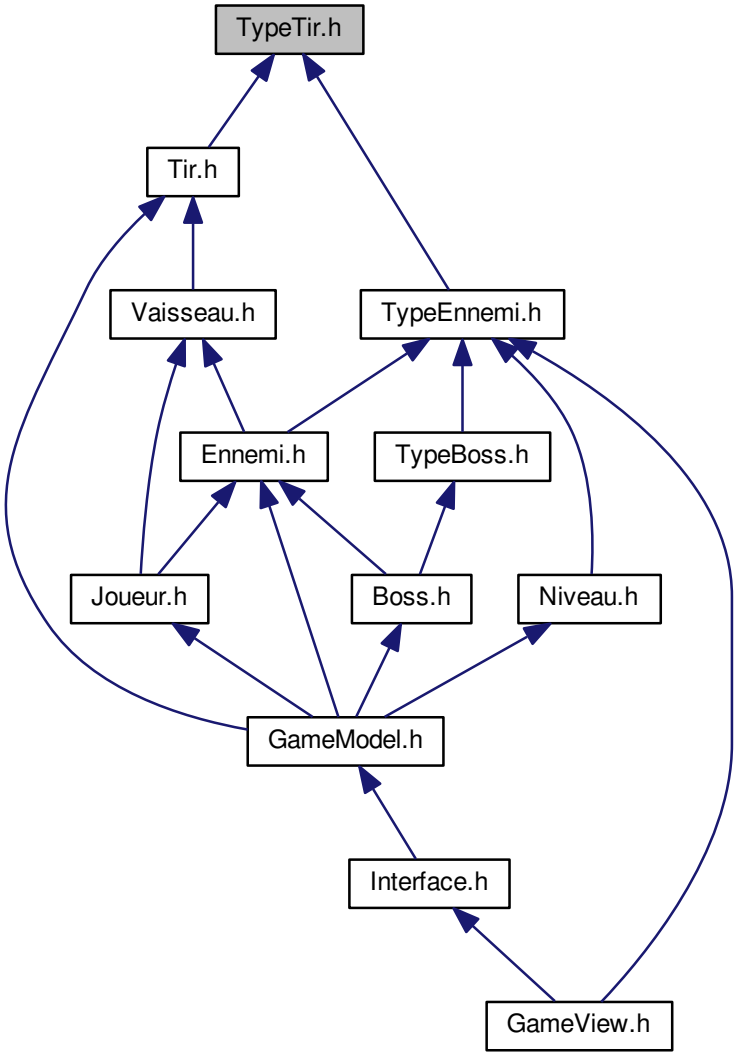
Déclaration de la classe TypeTir.

```
#include <string>
```

Graphe des dépendances par inclusion de TypeTir.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class TypeTir
Classe représentant les différents types de Tir.

6.17.1 Description détaillée

Déclaration de la classe TypeTir.

Auteur

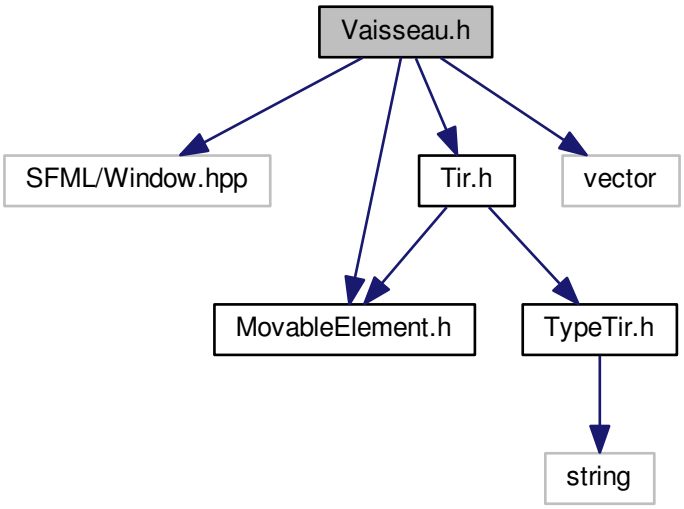
Quentin HARSCOËT

6.18 Référence du fichier Vaisseau.h

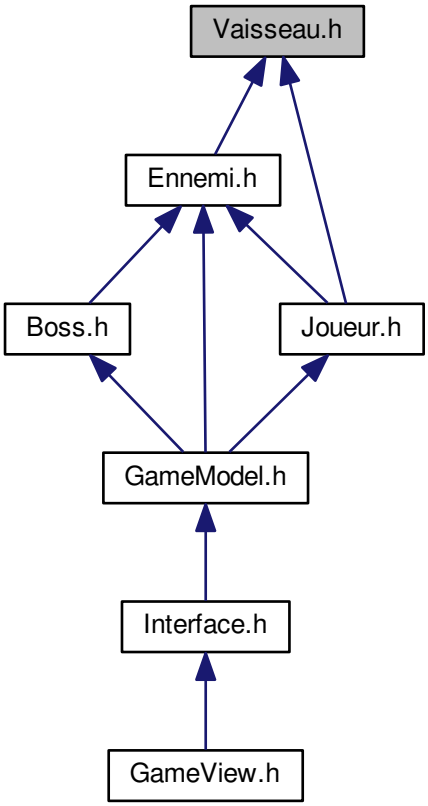
Déclaration de la classe Vaisseau.

```
#include <SFML/Window.hpp>
#include "MovableElement.h"
#include "Tir.h"
#include <vector>
```

Graphe des dépendances par inclusion de Vaisseau.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

– class Vaisseau
Classe représentant les vaisseaux.

6.18.1 Description détaillée

Déclaration de la classe Vaisseau.

Auteur

Théo CHASSAIGNE
Quentin HARSCOËT

Index