```
#include "Model.h"
#include "Menu.h"
#include "GameView.h"
#include "Settings.h"
#include "Ranking.h"
#include "Intro.h"
#include "Rules.h"

#include <iostream>

const int SCREEN_WIDTH = 1200;
const int SCREEN_HEIGHT = 600;

using namespace std;




//================================================================================
================================
// Description : FONCTION MERE
// Auteur : Guillaume Nedelec
// Date : 01/02/16
// InterÃªt : permet d'initialiser un modÃ¨le et des vues (GameView, Rules, Menu
...)c'est la fonction 'dÃ©clencheuse'
//================================================================================
================================
int main()
{
    srand(time(NULL));
    sf::RenderWindow window(sf::VideoMode(SCREEN_WIDTH, SCREEN_HEIGHT, 32), "Run
ner", sf::Style::Close);
    window.setFramerateLimit(60);

    Model model(SCREEN_WIDTH, SCREEN_HEIGHT);
    Intro* intro = new Intro(SCREEN_WIDTH,SCREEN_HEIGHT, &window);
    Menu* menu;
    GameView* game;
    Settings* settings;
    Ranking* ranking;
    Rules* rules;

    while(intro->treatEvents())
    {
        intro->draw();
        intro->synchronize();
    }
    delete intro;

    menu = new Menu(SCREEN_WIDTH, SCREEN_HEIGHT, &window);

    while(window.isOpen())
    {
        while(menu->treatEvents())
        {
            menu->draw();
            menu->synchronize();
        }

        if(menu->getStatut() == GAME)
        {
            game = new GameView(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
            delete menu;
            model.setEndGame(false);
            game->setModel(&model);

            while(game->treatEvents())
            {
```



```
                if(!game->getPause())
                    model.nextStep();
                game->synchronize();
                game->draw();
            }
            delete game;
            model.reset();
            menu = new Menu(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
        }

        else if(menu->getStatut() == SETTINGS)
        {
            settings = new Settings(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
            delete menu;
            settings->setModel(&model);
            while(settings->treatEvents())
            {
                settings->draw();
                settings->synchronize();
            }
            delete settings;
            menu = new Menu(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
        }

        else if(menu->getStatut() == RANKING)
        {
            ranking = new Ranking(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
            delete menu;

            while(ranking->treatEvents())
            {
                ranking->draw();
                ranking->synchronize();
            }
            delete ranking;
            menu = new Menu(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
        }

        else if(menu->getStatut() == RULES)
        {
            rules = new Rules(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
            delete menu;

            while(rules->treatEvents())
            {
                rules->draw();
                rules->synchronize();
            }
            delete rules;
            menu = new Menu(SCREEN_WIDTH, SCREEN_HEIGHT, &window);
        }
    }
    delete menu;
    return EXIT_SUCCESS;
}
```

```
#include "AnimatedGraphicElement.h"



//=========================================================================
========================================
// Description : Constructeur de AnimatedGraphicElement
// Auteur : Guillaume Nedelec
// Date : 8/04/16
// InterÃªt : permet de creer des AnimatedGraphicELement Ã  partir d'un vecteur
de lecture, d'une texture et de quatre entiers
//=========================================================================
========================================
AnimatedGraphicElement::AnimatedGraphicElement(const std::vector<sf::IntRect> &c
lipRects, sf::Texture &image, int x, int y, int w, int h) :
        GraphicElement{image,x,y,w,h}, _clip_rects{clipRects}
{}




//=================================================
// Description : Destructeur de AnimatedGraphicElement
// Auteur : Guillaume Nedelec
// Date : 8/04/16
// InterÃªt : permet de detruire l'objet
//=================================================
AnimatedGraphicElement::~AnimatedGraphicElement() {}




//=========================================================================
============================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 8/04/16
// InterÃªt : permet de dessiner sur une fenÃªtre, un Ã©lÃ©ment graphique qui ch
angera en fonction du temps (horloge)
//=========================================================================
============================
void AnimatedGraphicElement::draw(sf::RenderWindow *window)
{
    setTextureRect(_clip_rects[_current_clip_rect%_clip_rects.size()]);
    window->draw(*this);

    sf::Time time = _clockFrame.getElapsedTime();
    if(time.asSeconds() > 0.1)
    {
        _current_clip_rect++;
        _clockFrame.restart();
    }
}




//=========================================================================
====
// Description : Mutateur de vecteur de lecture
// Auteur : Nicolas Marcilloux
// Date : 24/05/16
// InterÃªt : permet d'appliquer un vecteur de lecture sur un Ã©lÃ©ment graphiqu
e animÃ©
//=========================================================================
```



```
====
void AnimatedGraphicElement::setVectRect(std::vector<sf::IntRect> c)
{
   _clip_rects = c;
}
```

```
#ifndef ANIMATEDGRAPHICELEMENT_H
#define ANIMATEDGRAPHICELEMENT_H

#include "GraphicElement.h"

//=========================================================================
// Description:
//
// Cette classe permet de creer et gerer des Ã©lÃ©ments graphiques animÃ©s
//=========================================================================

class AnimatedGraphicElement : public GraphicElement
{
protected:
    std::vector<sf::IntRect> _clip_rects; //Ensemble des rectangles de lectures
Ã  lire sur la feuille de sprites
    unsigned int _current_clip_rect = 0; //Indice du rectangle de lecture couran
t (qui est ou va Ãªtre affichÃ©)
    sf::Clock _clockFrame; //Horloge permettant de changer de rectangle de lectu
re au bout d'un certain temps

public:
    AnimatedGraphicElement(const std::vector<sf::IntRect> & clipRects, sf::Textu
re & image, int x, int y, int w, int h);
    virtual void draw(sf::RenderWindow *window) override;
    virtual ~AnimatedGraphicElement();
    void setVectRect(std::vector<sf::IntRect> c);
};

#endif // ANIMATEDGRAPHICELEMENT_H
```

```
#include "Bonus.h"




//=============================================================================
============================
// Description : Constructeur de Bonus
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de creer des Bonus Ã  partir de 4 float, 2 unsigned int, un
 BonusType et un HeightElement
//=============================================================================
============================
Bonus::Bonus(float x, float y, unsigned int w, unsigned int h, float dx, float d
y, BonusType type, HeightElement height):
    MovableElement{x,y,w,h,dx,dy}, _type{type}
{
     _height = height;
}




//====================================
// Description : Destructeur de Bonus
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de detruire l'objet
//====================================
Bonus::~Bonus() {}




//=============================================
// Description : Accesseur du type
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de rÃ©cupÃ©rer le type du Bonus
//=============================================
BonusType Bonus::get_type() const
{
    return _type;
}
```

```
#ifndef BONUS_H
#define BONUS_H

#include "MovableElement.h"

//DÃ©finit les differents types de bonus
enum BonusType
{
    Invincible,
    Health,
    Jump,
    Pointx2,
    Nuke
};

//=========================================================================
// Description:
//
// Cette classe permet de definir les propriÃ©tÃ©s des bonus ainsi que les effet
s qu'ils doivent produire.
//=========================================================================

class Bonus : public MovableElement
{
private:
    BonusType _type; //Type du bonus

public:
    Bonus(float x, float y, unsigned int w, unsigned int h, float dx, float dy,
BonusType type, HeightElement height);
    virtual ~Bonus() override;
    BonusType get_type() const;
};

#endif // BONUS_H
```

```
#include "Button.h"
#include "View.h"




//=========================================================================
=====
// Description : Constructeur de Button
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de creer des Button Ã  partir de 2 float, une couleur et un
 string
//=========================================================================
=====
Button::Button(float w, float h, sf::Color col, std::string text) :
     _w{w}, _h{h}, _color{col}
{
    _struct = sf::RectangleShape{sf::Vector2f{_w,_h}};
    _struct.setFillColor(_color);

    _text.setFont(View::_font);
    _text.setColor(sf::Color::White);
    _text.setString(text);
    _text.setPosition(_struct.getPosition().x, _struct.getPosition().y);
}




//====================================
// Description : Destructeur de Button
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de detruire l'objet
//====================================
Button::~Button(){}




//=========================================================================
// Description : Changement de couleur 1
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de changer la couleur (en paramÃ¨tre) du rectangle du bouto
n
//=========================================================================
void Button::rectColorChange(sf::Color col)
{
    _color = col;
    _struct.setFillColor(col);
}




//=========================================================================
// Description : Changement de couleur 2
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de changer la couleur (en paramÃ¨tre) du texte du bouton
//=========================================================================
void Button::textColorChange(sf::Color col)
{
```



```
    _textColor = col;
    _text.setColor(_textColor);
}




//=========================================================
// Description : Changement de taille de texte
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de modifier la police du texte du bouton
//=========================================================
void Button::textSize(unsigned int size)
{
    _text.setCharacterSize(size);
}




//=============================================================
// Description : Changement de taille de Rectangle
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de modifier la taille du rectangle du bouton
//=============================================================
void Button::rectSize(float w, float h)
{
    _struct = sf::RectangleShape{sf::Vector2f{w,h}};
}




//=========================================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de dessiner le bouton (rectangle et texte) sur une fenÃªtre
//=========================================================================
void Button::draw(sf::RenderWindow *window)
{
    window->draw(_struct);
    window->draw(_text);
}




//=============================================================
// Description : Mutateur de position rectangle
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de modifier la position du rectangle du bouton
//=============================================================
void Button::setPosition(int x, int y)
{
    _x = x;
    _y = y;
    _struct.setPosition(_x,_y);
}
```

```
//=========================================================
// Description : Mutateur de position texte
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de modifier la position du texte du bouton
//=========================================================
void Button::setTextPosition(int x, int y)
{
    _text.setPosition(x, y);
}




//=========================================================
// Description : Accesseur de sÃ©lÃ©ction
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de savoir si le bouton est sÃ©lÃ©ctionnÃ©
//=========================================================
bool Button::getSelected()
{
    return _selected;
}




//=========================================
// Description : Mutateur de sÃ©lÃ©ction
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de sÃ©lÃ©ctionnÃ© un bouton
//=========================================
void Button::setSelection(bool b)
{
    _selected = b;
}




//=============================================================
// Description : Accesseur de position horizontale
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de d'obtenir la position horizontale du bouton
//=============================================================
unsigned int Button::get_x()
{
    return _x;
}




//=========================================================
// Description : Accesseur de position verticale
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de d'obtenir la position verticale du bouton
//=========================================================
unsigned int Button::get_y()
```



```
{
    return _y;
}




//=================================================
// Description : Accesseur de largeur
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de d'obtenir la largeur du bouton
//=================================================
float Button::get_w()
{
    return _w;
}




//=================================================
// Description : Accesseur de longueur
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de d'obtenir la longueur du bouton
//=================================================
float Button::get_h()
{
    return _h;
}




//=================================================
// Description : Mutateur de texte
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de modifier le texte d'un bouton
//=================================================
void Button::setText(std::string text)
{
    _text.setString(text);
}
```

```
#ifndef BUTTON_H
#define BUTTON_H

#include <SFML/Graphics.hpp>

//============================================================================
// Description:
//
// Cette classe permet de gÃ©nÃ©rer et modifier des boutons crÃ©Ã©s Ã  partir d'
un rectangle et d'un texte.
//============================================================================
class Button
{
private :
    sf::RectangleShape _struct; //Structure du bouton
    float _w, _h; //Largeur et hauteur du bouton
    sf::Color _color = sf::Color::White; //Couleur du bouton
    sf::Color _textColor = sf::Color::White; //Couleur du texte dans le bouton
    sf::Text _text; //Texte du bouton
    bool _selected = false; //Indique si le curseur de la souris est sur le bout
on ou non
    int _x, _y =0; //position x et y du bouton


public:
    Button(float w, float h, sf::Color col, std::string text);
    ~Button();
    void rectColorChange(sf::Color col);
    void textColorChange(sf::Color col);
    void textSize(unsigned int size);
    void rectSize(float w, float h);
    void draw(sf::RenderWindow* window);
    bool getSelected();
    void setSelection(bool b);
    void setPosition(int x, int y);
    void setTextPosition(int x, int y);
    void setText(std::string text);
    float get_w();
    float get_h();
    unsigned int get_x();
    unsigned int get_y();
};

#endif // BUTTON_H
```

```
#include "Coin.h"




//=========================================================================
============================
// Description : Constructeur de Coin
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de creer des Coin Ã  partir de 4 float, 2 unsigned int et u
n HeightElement
//=========================================================================
============================
Coin::Coin(float x, float y, unsigned int w, unsigned int h, float dx, float dy,
 HeightElement height):
    MovableElement{x,y,w,h,dx,dy}
{
    _height = height;
}




//====================================
// Description : Destructeur de Coin
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de detruire l'objet
//====================================
Coin::~Coin() {}
```

```
#ifndef COINS_H
#define COINS_H

#include "MovableElement.h"

// Indique si les piÃ¨ces apparaissent par series de 3, 4 ou 5 piÃ¨ces.
enum TypeCoinSeries
{
    SeriesOf3,
    SeriesOf4,
    SeriesOf5
};

//=========================================================================
// Description:
//
// Cette classe permet de dÃ©finir les propriÃ©tÃ©s des piÃ¨ces Ã  recupÃ©rer da
ns le jeu
//=========================================================================

class Coin : public MovableElement
{
public:
    Coin(float x, float y, unsigned int w, unsigned int h, float dx, float dy, H
eightElement height);
    virtual ~Coin() override;
};

#endif // COINS_H
```

```
#include "GameView.h"
#include "Rules.h"
#include <iostream>
#include <fstream>

using namespace std;




//================================================================================
=======
// Description : Constructeur de GameView
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de construire une GameView Ã  partir d'une fenetre et de de
ux entier
//================================================================================
=======
GameView::GameView(int w, int h, sf::RenderWindow *window) :
    View{w,h,window}
{
    _statut = GAME;
    if(loadImages())
    {
        cuttingSprite();
        cleanSprite();
        loadSound();

        /***JEU ***/
        _bonusInvincible = new GraphicElement(_TextureBonusInvincible, 0.f, 0.f,
 SIZE_BONUS,SIZE_BONUS );
        _bonusLife = new GraphicElement(_TextureBonusLife, 0.f, 0.f, SIZE_BONUS,
SIZE_BONUS );
        _bonusDoubleJump = new GraphicElement(_TextureBonusJump, 0.f, 0.f, SIZE_
BONUS,SIZE_BONUS );
        _bonusDoublePoint = new GraphicElement(_TextureBonusPoint, 0.f, 0.f, SIZ
E_BONUS,SIZE_BONUS );
        _bonusNuke = new GraphicElement(_TextureBonusNuke, 0.f, 0.f, SIZE_BONUS,
SIZE_BONUS );

        _tree = new GraphicElement(_TextureTree, 0.f, 0.f, WIDTH_TREE,HEIGHT_TRE
E );
        _rock = new GraphicElement(_TextureRock, 0.f, 0.f, WIDTH_ROCK,HEIGHT_ROC
K );
        _eagle = new AnimatedGraphicElement(AnimEagle,_TextureEagle, 0.f, 0.f, W
IDTH_EAGLE,HEIGHT_EAGLE );
        _helico = new AnimatedGraphicElement(AnimHelico, _TextureHelico, 0, 0, 0
, 0);
        _bird = new AnimatedGraphicElement(AnimBird, _TextureBird, 0, 0, 0, 0);
        _wolf = new AnimatedGraphicElement(AnimWolf,_TextureWolf, 0.f, 0.f, WIDT
H_WOLF,HEIGHT_WOLF );

        _yeti = new AnimatedGraphicElement(AnimYetiForward, _TextureYeti, 0, 0,
0, 0);
        _yetiJump = new GraphicElement(_TextureYetiJump, 0, 0, 0, 0);
        _yetiJump2 = new GraphicElement(_TextureYetiJump2, 0, 0, 0, 0);
        _yetiFace = new AnimatedGraphicElement(AnimYetiStand, _TextureYetiFaces,
 5, 540, 0, 0);

        _lifebar = new GraphicElement(_TextureLifebar, 0, 530, 50, 100);
        _coin = new AnimatedGraphicElement(AnimCoin,_TextureSilverCoin, 0.f, 0.f
, SIZE_COIN,SIZE_COIN);

        _shield = new AnimatedGraphicElement(AnimShield, _TextureShield, 0, 0, 1
00, 100);
        _nuke = new AnimatedGraphicElement(AnimNuke, _TextureNuke, 500, 200, 0,
0);
        _impact = new AnimatedGraphicElement(AnimImpact, _TextureCollision, 0, 0
```



```
, 0, 0);
        _explosion = new AnimatedGraphicElement(AnimExplosion, _TextureExplosion
, 0, 0, 0, 0);

        _yeti->setOrigin(SIZE_PLAYER/2, SIZE_PLAYER/2);
        _shield->setColor(sf::Color(255,255,255,128));

        _soundCoins.setBuffer(_bufferCoin);
        _soundBonus.setBuffer(_bufferBonus);
        _soundHealthBonus.setBuffer(_bufferHealthBonus);
        _soundWolf.setBuffer(_bufferWolf);
        _soundEagle.setBuffer(_bufferEagle);
        _soundImpact.setBuffer(_bufferImpact);
        _soundNuke.setBuffer(_bufferNuke);
        _soundHelico.setBuffer(_bufferHelico);
        _soundHelico.setVolume(15);
        _soundCoins.setVolume(10);
        _soundEagle.setVolume(15);
        _soundImpact.setVolume(40);
        _soundHealthBonus.setVolume(20);
        _soundNuke.setVolume(40);


        _flashBomb = sf::RectangleShape(sf::Vector2f{(float)(w),(float)(h)});
        _flashBomb.setFillColor(sf::Color(255,255,255,_opacity));

        _delimitation.setFillColor(sf::Color::Black);
        _delimitation.setPosition(0,525);

        _score_distance.setFont(_font);
        _score_distance.setScale(1.5, 1.5);
        _score_distance.setPosition(900, 530);
        _score_distance.setColor(sf::Color::Black);

        _textPause.setFont(_font);
        _textPause.setColor(sf::Color::Black);
        _textPause.setStyle(sf::Text::Italic);

        _takenCoins.setFont(_font);
        _takenCoins.setPosition(810, 545);
        _takenCoins.setColor(sf::Color::Black);

        _lifeInfo.setFont(_font);
        _lifeInfo.setPosition(265, 562);
        _lifeInfo.setColor(sf::Color::Black);
        _lifeInfo.setScale(0.5, 0.5);

        _overText.setFont(_font);
        _overText.setColor(sf::Color::White);
        _overText.setString("GAME OVER");
        _overText.setCharacterSize(72);
        _overText.setPosition(400, 250);

        _highScores.setFont(_font);
        _highScores.setColor(sf::Color::White);
        _highScores.setCharacterSize(36);

        /*** PAUSE ***/

        _musicButton->setPosition(50,410);
        _soundButton->setPosition(50,500);

        _resume_game = new Button(300,75,sf::Color(18,140,225,230), _stringResum
e);
        _resume_game->setPosition(450,160);

        _rules = new Button(300,75,sf::Color(18,140,225,230), _stringRules);
        _rules->setPosition(450,260);
```

```
        _backToMenu = new Button(300,75,sf::Color(18,140,225,230), _stringBackto
Menu);
        _backToMenu->setPosition(450,360);
        _backToMenu->setTextPosition(500,375);

        _header = sf::RectangleShape{sf::Vector2f{600,60}};
        _header.setOutlineColor(sf::Color::Black);
        _header.setFillColor(sf::Color(18,140,225,150));
        _header.setOutlineThickness(7);
        _header.setPosition(300, 30);

        _pauseTitle.setFont(_fontTitle);
        _pauseTitle.setColor(sf::Color::Black);
        _pauseTitle.setString("PAUSE");
        _pauseTitle.setCharacterSize(48);
        _pauseTitle.setPosition(530, 28);
    }
}


//================================================================================
==========================================
// Description : Destructeur de GameView
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
uement dans le GameView et de supprimer ce dernier
//================================================================================
==========================================
GameView::~GameView()
{
    delete _lifebar;
    delete _coin;
    delete _tree;
    delete _rock;
    delete _wolf;
    delete _eagle;
    delete _bonusInvincible;
    delete _bonusLife;
    delete _bonusDoubleJump;
    delete _bonusDoublePoint;
    delete _bonusNuke;
    delete _shield;
    delete _resume_game;
    delete _rules;
    delete _backToMenu;
    delete _nuke;
    delete _impact;
    delete _yeti;
    delete _helico;
    delete _bird;
    delete _explosion;
    delete _yetiJump;
    delete _yetiJump2;
    delete _yetiFace;
}


//================================================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : Permet de dessiner tous les elements graphiques sur la fenÃªtre
//================================================================================
void GameView::draw()
{
    View::draw();
    if(_game && !_pause)
    {
```



```
        if(_model->getPlayer()->getLife()>0)
        {
            for(auto e : _model->getCoin())
            {
                int x,y;
                _model->getCoinPosition(e,x,y);
                _coin->setPosition(x,y);
                _coin->draw(_window);
            }
            for(auto e : _model->getBonus())
            {
                int x,y;
                _model->getBonusPosition(e,x,y);

                if(e->get_type() == Invincible)
                {
                    _bonusInvincible->setPosition(x,y);
                    _bonusInvincible->draw(_window);
                }
                else if(e->get_type() == Health)
                {
                    _bonusLife->setPosition(x,y);
                    _bonusLife->draw(_window);
                }
                else if(e->get_type() == Jump)
                {
                    _bonusDoubleJump->setPosition(x,y);
                    _bonusDoubleJump->draw(_window);
                }
                else if(e->get_type() == Pointx2)
                {
                    _bonusDoublePoint->setPosition(x,y);
                    _bonusDoublePoint->draw(_window);
                }
                else if(e->get_type() == Nuke)
                {
                    _bonusNuke->setPosition(x,y);
                    _bonusNuke->draw(_window);
                }
            }
            for(Obstacle* e : _model->getObstacle())
            {
                int x,y;

                _model->getObstaclePosition(e,x,y);

                if(e->get_type() == ObstacleType1)
                {

                    if(e->get_height() == Ground)
                    {
                        _rock->setPosition(x-30,y);
                        _rock->draw(_window);
                    }
                    else
                    {
                        _bird->setPosition(x-30,y);
                        _bird->draw(_window);
                    }


                }
                else if(e->get_type() == ObstacleType2)
                {
                    if(e->get_height() == Ground)
                    {
                        _tree->setPosition(x-50,y);
                        _tree->draw(_window);
                    }
```

```
                    else
                    {

                        _eagle->setPosition(x-30,y);
                        _eagle->draw(_window);
                    }
                }

                else if(e->get_type() == ObstacleType3)
                {
                    if(e->get_height() == Ground)
                    {
                        _wolf->setPosition(x-30,y);
                        _wolf->draw(_window);
                    }
                    else
                    {
                        _helico->setPosition(x-30,y);
                        _helico->draw(_window);
                    }
                }

            }

            if(_colStaticElement)
            {
                if(_colTree)
                    _impact->setPosition(_col_x - 30, _col_y + 40);
                else
                    _impact->setPosition(_col_x, _col_y);
                _impact->draw(_window);
                sf::Time t = _clockStaticElem.getElapsedTime();

                _col_x += _model->getSpeedElement();

                if(t.asSeconds() > 1)
                    _colStaticElement = false;
            }

            if(_colMovingElement)
            {
                _impact->setPosition(_posXElem, _posYElem);
                _impact->draw(_window);
                sf::Time t3 = _clockMovingElem.getElapsedTime();

                _posXElem += _model->getSpeedElement();

                if(t3.asSeconds() > 1)
                    _colMovingElement = false;
            }

            if(_colHelico)
            {
                _explosion->setPosition(_posXCopter, _posYCopter);
                _explosion->draw(_window);
                sf::Time t2 = _clockCopter.getElapsedTime();
                _posXCopter += _model->getSpeedElement();
                if(t2.asSeconds() > 2.5)
                    _colHelico = false;
            }

            if(_nukeActivation)
            {
                _nuke->draw(_window);
                _nuke->setPosition((_nuke->getPosition().x + _model->getSpeedEle
ment()), _nuke->getPosition().y);
                _window->draw(_flashBomb);

                if(!_endNukeFlash)
```



```
                {
                    if(_opacity < 245)
                        _opacity += 10;
                    else if(_opacity < 255 && _opacity >= 245)
                    {
                        _opacity = 255;
                        _endNukeFlash = true;
                    }
                }
                else
                {
                    if(_opacity >= 10)
                        _opacity-= 10;
                    else
                        _opacity = 0;
                }


                _flashBomb.setFillColor(sf::Color(255,255,255,_opacity));

                sf::Time time = _clockNuke.getElapsedTime();

                if(time.asSeconds() > 2.192)
                {
                    _nukeActivation = false;
                    _endNukeFlash = false;
                    _opacity = 0;
                }
            }

            _window->draw(_delimitation);
            if((_colStaticElement || _colHelico || _colMovingElement) && !_invin
cible)
                _yetiFace->setVectRect(AnimYetiHit);
            else
                _yetiFace->setVectRect(AnimYetiStand);

            _yetiFace->draw(_window);
            _window->draw(_lifeLevel);
            _window->draw(_lifeInfo);
            _lifebar->draw(_window);
            _window->draw(_score_distance);
            _window->draw(_takenCoins);
            _coin->setPosition(750, 540);
            _coin->draw(_window);

            if(_model->getDoublePointBonus())
            {
                _bonusDoublePoint->setPosition(600,540);
                _bonusDoublePoint->draw(_window);
                _coin->setTexture(_TextureGoldCoin);
            }
            else
            {
                _coin->setTexture(_TextureSilverCoin);
            }

            if(_jump)
            {
                if(_left)
                    _yetiJump2->draw(_window);
                else
                    _yetiJump->draw(_window);
            }
            else
                _yeti->draw(_window);

            if(_model->getPlayer()->getInvincibleBonus())
            {
```

```
                _bonusInvincible->setPosition(655,540);
                _bonusInvincible->draw(_window);
                _shield->draw(_window);
            }

            if(_model->getPlayer()->getDoubleJumpBonus())
            {
                _bonusDoubleJump->setPosition(545,540);
                _bonusDoubleJump->draw(_window);
            }


            _window->draw(_textPause);
        }
    }
    else if(_pause)
    {
        _musicButton->draw(_window);
        _soundButton->draw(_window);
        _window->draw(_header);
        _window->draw(_pauseTitle);
        _resume_game->draw(_window);
        _rules->draw(_window);
        _backToMenu->draw(_window);
        _cursor->draw(_window);
    }
    else if(!_game)
    {
        _soundCoins.stop();
        _soundBonus.stop();
        _soundHealthBonus.stop();
        _soundWolf.stop();
        _soundEagle.stop();
        _soundImpact.stop();
        _soundNuke.stop();
        _soundHelico.stop();
        sf::Time time = _clock.getElapsedTime();
        if(time.asSeconds() < 2)
        {
            _window->clear(sf::Color::Black);
            _window->draw(_overText);
            if(checkScore())
                _window->draw(_highScores);
        }
        else
        {
            _gameOver = true;
            _model->setEndGame(true);
        }
    }
    _window->display();
}


//=========================================================================
==========================================================================
// Description : Synchronisation avec le modÃ¨le et le traitement d'Ã©venements
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de rÃ©cupÃ©rer en temps rÃ©el toutes les infos du modÃ¨le l
es evenements provoquÃ©s par l'utilisateur et de mettre Ã  jour l'interface grap
hique
//=========================================================================
==========================================================================
void GameView::synchronize()
{
    View::synchronize();

    _pauseTitle.setString("PAUSE");
```



```
    if(_lang == English)
    {
        _stringTextPause = "Press 'Espace' to Pause";
        _stringResume = "Resume Game";
        _stringRules = "Rules";
        _stringBacktoMenu = "Back To Menu";
        _stringHighScores = "You enter the Highscore !";

        _textPause.setPosition(950, 5);
        _textPause.setCharacterSize(20);
        _resume_game->setTextPosition(500,175);
        _rules->setTextPosition(560,275);
        _backToMenu->setTextPosition(500,375);
        _highScores.setPosition(375, 375);
    }
    else if(_lang == French)
    {
        _stringTextPause = "Appuyer sur 'Echap' pour faire pause";
        _stringResume = "Reprendre la partie";
        _stringRules = "Regles du jeu";
        _stringBacktoMenu = "Retour au Menu";
        _stringHighScores = "Vous entrez dans les meilleurs scores !";

        _textPause.setPosition(815, 5);
        _textPause.setCharacterSize(20);
        _resume_game->setTextPosition(460,175);
        _rules->setTextPosition(505,275);
        _backToMenu->setTextPosition(490,375);
        _highScores.setPosition(300, 375);
    }
    else if(_lang == Spanish)
    {
        _stringTextPause = "Pulse 'Escape' para pausar";
        _stringResume = "Continuar partida";
        _stringRules = "Reglas del juego";
        _stringBacktoMenu = "Volver al Menu";
        _pauseTitle.setString("PAUSA");
        _stringHighScores = "Se introduce la puntuacion mas alta! !";

        _textPause.setPosition(915, 5);
        _textPause.setCharacterSize(20);
        _resume_game->setTextPosition(475,175);
        _rules->setTextPosition(485,275);
        _backToMenu->setTextPosition(500,375);
        _highScores.setPosition(300, 375);
    }
    else if(_lang == German)
    {
        _stringTextPause = "Drucken Sie die 'Flucht' zu Pause";
        _stringResume = "Fortsetzen der Wiedergabe";
        _stringRules = "Regeln des Spiels";
        _stringBacktoMenu = "Zuruck zum Menu";
        _stringHighScores = "Sie geben den HighScore !";

        _textPause.setPosition(865, 5);
        _textPause.setCharacterSize(20);
        _resume_game->setTextPosition(460,180);
        _rules->setTextPosition(515,280);
        _backToMenu->setTextPosition(505,380);
        _highScores.setPosition(400, 375);

        _resume_game->textSize(22);
        _rules->textSize(22);
        _backToMenu->textSize(22);
    }

    _textPause.setString(_stringTextPause);
    _resume_game->setText(_stringResume);
    _rules->setText(_stringRules);
```

```
    _backToMenu->setText(_stringBacktoMenu);
    _highScores.setString(_stringHighScores);

    if(!_pause)
    {
        if(!_model->getPlayer()->getJump())
        {
            _doubleJumpBonusActivated = false;
            _jump = false;
        }
        else
            _jump = true;

        _model->scoreCalculation();
        _scoreView = _model->getScore();

        for(auto e : _model->getObstacle())
        {

            if (_model->getPlayer()->collision(e))
            {
                if(e->get_type() == ObstacleType3 && e->get_height() == Up_Air)
                {
                    _colHelico = true;
                    _model->getObstaclePosition(e,_posXCopter,_posYCopter);
                    _clockCopter.restart();
                }
                else if(e->get_type() != ObstacleType3 && e->get_height() == Gro
und)
                {
                    if(e->get_type() == ObstacleType2)
                        _colTree = true;
                    else
                        _colTree = false;

                    _colStaticElement = true;
                    _model->getObstaclePosition(e, _col_x, _col_y);
                    _clockStaticElem.restart();
                }

                else
                {
                    _colMovingElement = true;
                    _model->getObstaclePosition(e,_posXElem,_posYElem);
                    _clockMovingElem.restart();
                }
            }
        }

        for(auto e : _model->getBonus())
        {
            if(e->get_type() == Nuke && _model->getPlayer()->collision(e))
            {
                _nuke->setPosition(500, 200);
                _clockNuke.restart();
                _nukeActivation = true;
            }
        }


        if(_sounds)
        {
            for(auto e : _model->getCoin())
            {
                if(_model->getPlayer()->collision(e))
                {
                    _soundCoins.play();
                }
            }
```



```
            for(auto e : _model->getBonus())
            {
                if(_model->getPlayer()->collision(e))
                {
                    if(e->get_type() == Health)
                        _soundHealthBonus.play();
                    else if (e->get_type() == Nuke)
                        _soundNuke.play();
                    else
                        _soundBonus.play();
                }
            }

            for(auto e : _model->getObstacle())
            {
                if(_model->getPlayer()->collision(e))
                {
                    _yetiFace->setVectRect(AnimYetiHit);
                    if(e->get_height() == Ground)
                    {
                        if(e->get_type() == ObstacleType3)
                            _soundWolf.play();
                        else
                            _soundImpact.play();
                    }

                    else if(e->get_height() == Up_Air)
                    {
                        if(e->get_type() == ObstacleType3)
                            _soundHelico.play();
                        else
                            _soundEagle.play();
                    }
                }
                else
                    _yetiFace->setVectRect(AnimYetiStand);
            }
        }

        _score_distance.setString("Score : " +std::to_string(_scoreView));
        _takenCoins.setString("x " + std::to_string(_model->getTakenCoins()));

        int x,y;
        _model->getPlayerPosition(x,y);
        _yeti->setPosition(x, y);
        _yetiJump->setPosition(x-20,y-50);
        _yetiJump2->setPosition(x-20,y-50);

        if(_model->getPlayer()->getInvincibleBonus())
        {
            _invincible = true;
            if(_jump)
            {
                _shield->setVectRect(AnimShield2);
                _shield->setPosition(x-60, y-50);
            }
            else
            {
                _shield->setVectRect(AnimShield);
                _shield->setPosition(x-70, y-60);
            }
        }
        else
            _invincible = false;

        if(_left)
            _yeti->setVectRect(AnimYetiBackward);
        else
```

```
            _yeti->setVectRect(AnimYetiForward);


        if(_model->getPlayer()->getLife() >0)
        {
            _lifeLevel= sf::RectangleShape{sf::Vector2f{(float)(_model->getPlaye
r()->getLife()*4),25}};
            _lifeInfo.setString(std::to_string(_model->getPlayer()->getLife()) +
 "%");

            if(_model->getPlayer()->getLife() >=60)
            {
                _lifeLevel.setFillColor(sf::Color::Green);
            }
            else if(_model->getPlayer()->getLife() >=20)
            {
                _lifeLevel.setFillColor(sf::Color::Yellow);
            }
            else
                _lifeLevel.setFillColor(sf::Color::Red);
        }

        else if(_model->getPlayer()->getLife() <= 0 && _game)
        {
            _game = false;
            _lifeLevel= sf::RectangleShape{sf::Vector2f{0,30}};
            _lifeInfo.setString("0%");
            _clock.restart();
        }
        _lifeLevel.setPosition(75,555);
    }

    else
    {
        if(_resume_game->getSelected())
            _resume_game->rectColorChange(sf::Color(18,225,225,180));
        else
            _resume_game->rectColorChange(sf::Color(18,140,225,230));
        if(_rules->getSelected())
            _rules->rectColorChange(sf::Color(18,225,225,180));
        else
            _rules->rectColorChange(sf::Color(18,140,225,230));
        if(_backToMenu->getSelected())
            _backToMenu->rectColorChange(sf::Color(18,225,225,180));
        else
            _backToMenu->rectColorChange(sf::Color(18,140,225,230));

        _cursor->setPosition(_mouseX,_mouseY);
    }

}



//=============================================================================
=======================
// Description : Detecteur et traitement d'Ã©venements
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les Ã©venements de l'utilisateur et de les t
raiter en fonction de ceux-ci
//=============================================================================
=======================
bool GameView::treatEvents()
{
    bool result = false;
    if(_window->isOpen())
    {
        result = true;
```



```
        if(_gameOver)
        {
            result = false;
            _statut = MENU;
        }

        _window_position = (_window->getPosition());
        _mouse_position = (sf::Mouse::getPosition());
        _mouseX = ((_mouse_position.x) - (_window_position.x)) -20;
        _mouseY = ((_mouse_position.y) - (_window_position.y)) +35;

        sf::Event event;
        while (_window->pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
            {
                _window->close();
                result = false;
            }
            if(_pause)
            {
                if(_mouseX >= _resume_game->get_x()-17 && _mouseX <= _resume_gam
e->get_x()+_resume_game->get_w()-15
                        && _mouseY >= _resume_game->get_y()-7 && _mouseY <= _res
ume_game->get_y()+_resume_game->get_h())
                    _resume_game->setSelection(true);

                else
                    _resume_game->setSelection(false);

                if(_mouseX >= _rules->get_x()-17 && _mouseX <= _rules->get_x()+_
rules->get_w()-15
                        && _mouseY >= _rules->get_y()-7 && _mouseY <= _rules->ge
t_y()+_rules->get_h())
                    _rules->setSelection(true);

                else
                    _rules->setSelection(false);

                if(_mouseX >= _backToMenu->get_x()-17 && _mouseX <= _backToMenu-
>get_x()+_backToMenu->get_w()-15
                        && _mouseY >= _backToMenu->get_y()-7 && _mouseY <= _back
ToMenu->get_y()+_backToMenu->get_h())
                    _backToMenu->setSelection(true);

                else
                    _backToMenu->setSelection(false);


                if(event.type == sf::Event::MouseButtonPressed && event.mouseBut
ton.button == sf::Mouse::Left)
                {
                    sf::Vector2f blocPosition = _musicButton->getPosition();
                    sf::Vector2f blocSoundPosition = _soundButton->getPosition()
;

                    if(_backToMenu->getSelected())
                    {
                        result = false;
                        _statut = MENU;
                    }
                    else if(_resume_game->getSelected())
                    {
                        _pause=false;
                    }
                    else if(_rules->getSelected())
                    {
                        Rules* rules = new Rules(_w,_h,_window);
                        while(rules->treatEvents())
```

```
                        {
                            rules->draw();
                            rules->synchronize();
                        }
                        delete rules;
                    }
                    else if(_mouseX >= blocPosition.x-20 &&  _mouseX <= blocPosi
tion.x+_musicButton->get_w()-40
                        && _mouseY >= blocPosition.y && _mouseY <= blocPosit
ion.y+_musicButton->get_h()-20)
                    {
                        if(_music.getStatus() == _music.Playing)
                            _music.pause();
                        else
                            _music.play();
                    }
                    else if(_mouseX >= blocSoundPosition.x-20 &&  _mouseX <= blo
cSoundPosition.x+_soundButton->get_w()-40
                        && _mouseY >= blocSoundPosition.y && _mouseY <= bloc
SoundPosition.y+_soundButton->get_h()-20)
                    {
                        if(_sounds)
                            _sounds = false;
                        else
                            _sounds = true;
                    }
                    _cursor->setTexture(_TextureCursorClicPressed);
                }

                else if(event.type == sf::Event::MouseButtonReleased && event.mo
useButton.button == sf::Mouse::Left)
                {
                    _cursor->setTexture(_TextureCursorClicReleased);
                }

                if(event.type == sf::Event::KeyPressed && event.key.code == sf::
Keyboard::Escape)
                {
                    if(_pause)
                        _pause = false;
                    else
                        _pause = true;
                }
            }
            else if(!_pause && _game)
            {

                if(event.type == sf::Event::KeyPressed)
                {
                    if(event.key.code == sf::Keyboard::Left)
                    {
                        if(_game && !_pause)
                        {
                            _model->getPlayer()->setLeft(true);
                            _model->getPlayer()->setRight(false);
                            _left = true;
                        }
                    }

                    if(event.key.code == sf::Keyboard::Escape)
                    {
                        if(_pause)
                            _pause = false;
                        else
                            _pause = true;
                    }

                    else if(event.key.code == sf::Keyboard::Right)
                    {
```



```
                        if(_game && !_pause)
                        {
                            _model->getPlayer()->setRight(true);
                            _model->getPlayer()->setLeft(false);
                            _left = false;
                        }
                    }
                    else if(event.key.code == sf::Keyboard::Up)
                    {
                        if(_game && !_pause)
                        {
                            if(_model->getPlayer()->getJump() && _model->getPlay
er()->getDoubleJumpBonus())
                            {
                                _model->getPlayer()->setDoubleJump(true);
                                if(!_doubleJumpBonusActivated)
                                {
                                    _model->getPlayer()->set_y(_model->getPlayer
()->get_y());
                                    _doubleJumpBonusActivated = true;
                                }
                            }

                            if(!_model->getPlayer()->getJump())
                                _model->getPlayer()->setJump(true);
                        }
                    }
                }
                if(event.type == sf::Event::KeyReleased)
                {
                    if(event.key.code == sf::Keyboard::Left)
                        _model->getPlayer()->setLeft(false);

                    else if(event.key.code == sf::Keyboard::Right)
                        _model->getPlayer()->setRight(false);
                }
            }
        }
    }
    return result;
}


//=========================================================================
==========================================
// Description : Decoupage de sprite
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de dÃ©couper les feuilles de sprites en plusieurs images af
in de les insÃ©rer dans des vecteurs de lecture
//=========================================================================
==========================================
void GameView::cuttingSprite()
{
    for(int i=0; i<12;i++)
    {
        sf::IntRect r(52*i,0,50,50);
        AnimCoin.push_back(r);
    }

    for(int i=0; i<6;i++)
    {
        sf::IntRect r8(150*i,0,150,125);
        AnimShield.push_back(r8);
    }

    for(int i=0; i<4;i++)
    {
        sf::IntRect r9(80*i,0,78,50);
```

```
        AnimBird.push_back(r9);
    }

    for(int i=0; i<7;i++)
    {
        sf::IntRect r10(150*i,0,148,75);
        AnimWolf.push_back(r10);
    }

    for(int i=0; i<4;i++)
    {
        sf::IntRect r11(100*i,0,99,50);
        AnimEagle.push_back(r11);
    }

    for(int i=0; i<23; i++)
    {
        sf::IntRect r12(300*i,0,300,300);
        AnimNuke.push_back(r12);
    }

    for(int i=0; i<3; i++)
    {
        sf::IntRect r13(67*i,0,67,67);
        AnimImpact.push_back(r13);
    }

    for(int i=0; i<4; i++)
    {
        sf::IntRect r14(75*i,0,73,75);
        AnimYetiForward.push_back(r14);
    }

    for(int i=0; i<4; i++)
    {
        sf::IntRect r15(75*i,75,73,75);
        AnimYetiBackward.push_back(r15);
    }

    for(int i=0; i<6; i++)
    {
        sf::IntRect r16(110*i,0,110,100);
        AnimHelico.push_back(r16);
    }

    for(int i=0; i<27; i++)
    {
        sf::IntRect r17(100*i,0,100,100);
        AnimExplosion.push_back(r17);
    }

    for(int i=0; i<6;i++)
    {
        sf::IntRect r18(150*i,125,150,150);
        AnimShield2.push_back(r18);
    }

    for(int i=0; i<3; i++)
    {
        sf::IntRect r19(60*i,0,60,50);
        AnimYetiStand.push_back(r19);
    }

    sf::IntRect r20(0, 55, 60, 50);
    AnimYetiHit.push_back(r20);
}

//=============================================================================
==
```



```
// Description : Mutateur de modÃ¨le
// Auteur : Guillaume Nedelec
// Date : 20/05/16
// InterÃªt : permet de charger de le modÃ¨le courant (celui construit dans le m
ain)
//=============================================================================
==
void GameView::setModel(Model * model)
{
    _model = model;
}



//=============================================================================
==========================================================
// Description : Chargement d'images
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de charger les images dans les textures et fait planter le
jeu si une image ne s'est pas chargÃ© (accompagnÃ© d'un message d'erreur)
//=============================================================================
==========================================================
bool GameView::loadImages()
{
    bool success = true;

    if (!_TextureYetiFaces.loadFromFile(AVATAR))
    {
        std::cerr << "ERROR when loading image file: " << AVATAR << std::endl;
        success = false;
    }
    if (!_TextureExplosion.loadFromFile(EXPLOSION))
    {
        std::cerr << "ERROR when loading image file: " << EXPLOSION << std::endl
;
        success = false;
    }
    if (!_TextureYeti.loadFromFile(YETI_WALK))
    {
        std::cerr << "ERROR when loading image file: " << YETI_WALK << std::endl
;
        success = false;
    }
    if (!_TextureYetiJump.loadFromFile(YETI_JUMP))
    {
        std::cerr << "ERROR when loading image file: " << YETI_JUMP << std::endl
;
        success = false;
    }
    if (!_TextureYetiJump2.loadFromFile(YETI_JUMP2))
    {
        std::cerr << "ERROR when loading image file: " << YETI_JUMP2 << std::end
l;
        success = false;
    }
    if (!_TextureBird.loadFromFile(BIRD))
    {
        std::cerr << "ERROR when loading image file: " << BIRD << std::endl;
        success = false;
    }
    if (!_TextureHelico.loadFromFile(HELICO))
    {
        std::cerr << "ERROR when loading image file: " << HELICO << std::endl;
        success = false;
    }
    if (!_TextureCollision.loadFromFile(IMPACT))
    {
        std::cerr << "ERROR when loading image file: " << IMPACT << std::endl;
```

```
        success = false;
    }
    if (!_TextureBonusNuke.loadFromFile(BONUS_NUKE))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_NUKE << std::end
l;
        success = false;
    }
    if (!_TextureEagle.loadFromFile(EAGLE))
    {
        std::cerr << "ERROR when loading image file: " << EAGLE << std::endl;
        success = false;
    }
    if (!_TextureWolf.loadFromFile(WOLF))
    {
        std::cerr << "ERROR when loading image file: " << WOLF << std::endl;
        success = false;
    }
    if (!_TextureLifebar.loadFromFile(LIFEBAR))
    {
        std::cerr << "ERROR when loading image file: " << LIFEBAR << std::endl;
        success = false;
    }
    if (!_TextureRock.loadFromFile(ROCK))
    {
        std::cerr << "ERROR when loading image file: " << ROCK << std::endl;
        success = false;
    }
    if (!_TextureShield.loadFromFile(SHIELD))
    {
        std::cerr << "ERROR when loading image file: " << SHIELD << std::endl;
        success = false;
    }
    if (!_TextureTree.loadFromFile(TREE))
    {
        std::cerr << "ERROR when loading image file: " << TREE << std::endl;
        success = false;
    }
    if (!_TextureNuke.loadFromFile(NUKE))
    {
        std::cerr << "ERROR when loading image file: " << NUKE << std::endl;
        success = false;
    }
    if (!_TextureSilverCoin.loadFromFile(SILVER_COIN))
    {
        std::cerr << "ERROR when loading image file: " << SILVER_COIN << std::en
dl;
        success = false;
    }
    if (!_TextureGoldCoin.loadFromFile(GOLD_COIN))
    {
        std::cerr << "ERROR when loading image file: " << GOLD_COIN << std::endl
;
        success = false;
    }
    if (!_TextureBonusLife.loadFromFile(BONUS_LIFE))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_LIFE << std::end
l;
        success = false;
    }
    if (!_TextureBonusInvincible.loadFromFile(BONUS_INVINCIBLE))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_INVINCIBLE << st
d::endl;
        success = false;
    }
    if (!_TextureBonusPoint.loadFromFile(BONUS_POINT))
    {
```



```
        std::cerr << "ERROR when loading image file: " << BONUS_POINT << std::en
dl;
        success = false;
    }
    if (!_TextureBonusJump.loadFromFile(BONUS_JUMP))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_JUMP << std::end
l;
        success = false;
    }
    return success;
}



//============================================================================
============================================================================
// Description : Chargement de sons
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de charger les sons dans les buffers et fait planter le jeu
 si une image ne s'est pas chargÃ© (accompagnÃ© d'un message d'erreur)
//============================================================================
============================================================================
void GameView::loadSound()
{
    if (!_bufferCoin.loadFromFile(SOUND_COIN))
        std::cerr << "ERROR when loading sound file: " << SOUND_COIN << std::end
l;

    if (!_bufferBonus.loadFromFile(SOUND_BONUS))
        std::cerr << "ERROR when loading sound file: " << SOUND_BONUS << std::en
dl;

    if (!_bufferHealthBonus.loadFromFile(SOUND_HEALTHBONUS))
        std::cerr << "ERROR when loading sound file: " << SOUND_HEALTHBONUS << s
td::endl;

    if (!_bufferWolf.loadFromFile(SOUND_WOLF))
        std::cerr << "ERROR when loading sound file: " << SOUND_WOLF << std::end
l;

    if (!_bufferEagle.loadFromFile(SOUND_EAGLE))
        std::cerr << "ERROR when loading sound file: " << SOUND_EAGLE << std::en
dl;

    if (!_bufferNuke.loadFromFile(SOUND_NUKE))
        std::cerr << "ERROR when loading sound file: " << SOUND_NUKE << std::end
l;

    if (!_bufferHelico.loadFromFile(SOUND_HELICO))
        std::cerr << "ERROR when loading sound file: " << SOUND_HELICO << std::e
ndl;

    if (!_bufferImpact.loadFromFile(SOUND_IMPACT))
        std::cerr << "ERROR when loading sound file: " << SOUND_IMPACT << std::e
ndl;
}

bool GameView::getPause()
{
    return _pause;
}



//============================================================================
============================================================================
// Description : Nettoyage des sprite
```

```
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de lisser les sprites chargÃ©s dans les textures
//=========================================================================
=========================================================================
void GameView::cleanSprite()
{
    _TextureTree.setSmooth(true);
    _TextureRock.setSmooth(true);
    _TextureWolf.setSmooth(true);
    _TextureEagle.setSmooth(true);
    _TextureNuke.setSmooth(true);
    _TextureSilverCoin.setSmooth(true);
    _TextureGoldCoin.setSmooth(true);
    _TextureBonusLife.setSmooth(true);
    _TextureBonusJump.setSmooth(true);
    _TextureBonusInvincible.setSmooth(true);
    _TextureBonusPoint.setSmooth(true);
    _TextureBonusNuke.setSmooth(true);
    _TextureShield.setSmooth(true);
    _TextureLifebar.setSmooth(true);
    _TextureCollision.setSmooth(true);
    _TextureYeti.setSmooth(true);
    _TextureHelico.setSmooth(true);
    _TextureBird.setSmooth(true);
    _TextureExplosion.setSmooth(true);
    _TextureYetiJump.setSmooth(true);
    _TextureYetiJump2.setSmooth(true);
}



//=========================================================================
==================================
// Description : VÃ©rification du score
// Auteur : Guillaume Nedelec
// Date : 20/05/16
// InterÃªt : permet de vÃ©rifier si le score atteind par le joueur est supÃ©rie
ur Ã  l'un prÃ©sent dans le classement
//=========================================================================
==================================
bool GameView::checkScore()
{
    fstream f;
    int element;
    string fichier = "scores.txt";
    bool enterToHighScore = false;

    std::ifstream infile(fichier);

    if(infile.good())
    {
        f.open(fichier.c_str(), ios:: in); //Ouverture en lecture

        if(f.fail())
        {
            std::cerr << "ouverture en lecture impossible" << endl;
            exit(EXIT_FAILURE);
        }
        while(!f.eof() && !enterToHighScore)
        {
            f >> element;
            if(_model->getScore() > (unsigned int)(element))
            {
                enterToHighScore = true;
            }
        }
        f.close();
    }
```



```
    return enterToHighScore;
}
```

```
#ifndef GAMEVIEW_H
#define GAMEVIEW_H

#include "View.h"
#include "Model.h"
#include "Button.h"

// Liens vers les fichiers images utilisÃ©s dans GameView et dans Rules

const std::string SILVER_COIN = "Images/MovableElement/Coins/SilverCoins.png";
const std::string GOLD_COIN = "Images/MovableElement/Coins/GoldCoins.png";
const std::string BONUS_LIFE = "Images/MovableElement/Bonus/BonusHealth.png";
const std::string BONUS_POINT = "Images/MovableElement/Bonus/BonusDoublePoint.pn
g";
const std::string BONUS_INVINCIBLE = "Images/MovableElement/Bonus/BonusInvincibl
e.png";
const std::string BONUS_JUMP = "Images/MovableElement/Bonus/BonusDoubleJump.png"
;
const std::string BONUS_NUKE = "Images/MovableElement/Bonus/BonusNuke.png";

const std::string TREE = "Images/MovableElement/Obstacles/tree.png";
const std::string ROCK = "Images/MovableElement/Obstacles/rock.png";
const std::string WOLF = "Images/MovableElement/Obstacles/wolves.png";
const std::string EAGLE = "Images/MovableElement/Obstacles/eagle.png";
const std::string HELICO = "Images/MovableElement/Obstacles/helico.png";
const std::string BIRD = "Images/MovableElement/Obstacles/bird.png";

const std::string YETI_WALK = "Images/MovableElement/Yeti/YetiWalk.png";


//=========================================================================
// Description:
//
// Cette classe reprÃ©sente la partie graphique du jeu
// Elle recupÃ¨re les informations du modÃ¨le pour les retranscrire Ã  l'Ã©cran
//=========================================================================
class GameView : public View
{
private :

    // Liens vers les fichiers images (.png) utilisÃ©s dans GameView
    const std::string EXPLOSION = "Images/MovableElement/Obstacles/Explosion.png
";
    const std::string SHIELD = "Images/MovableElement/Yeti/Shield.png";
    const std::string LIFEBAR = "Images/LifeBar/LifeBar.png";
    const std::string IMPACT = "Images/MovableElement/Obstacles/explosion.png";
    const std::string YETI_JUMP = "Images/MovableElement/Yeti/YetiJump.png";
    const std::string YETI_JUMP2 = "Images/MovableElement/Yeti/YetiJump2.png";
    const std::string AVATAR = "Images/MovableElement/Yeti/yetiFaces.png";
    const std::string NUKE = "Images/MovableElement/Bonus/Nuke.png";

    // Liens vers les fichiers sons (.wav) utilisÃ©s dans GameView
    const std::string SOUND_COIN = "Sounds/coin.wav";
    const std::string SOUND_BONUS = "Sounds/Bonus.wav";
    const std::string SOUND_HEALTHBONUS = "Sounds/HealthBonus.wav";
    const std::string SOUND_WOLF = "Sounds/wolf.wav";
    const std::string SOUND_EAGLE = "Sounds/eagle.wav";
    const std::string SOUND_NUKE = "Sounds/nuke.wav";
    const std::string SOUND_HELICO = "Sounds/helico.wav";
    const std::string SOUND_IMPACT = "Sounds/impact.wav";


    Model * _model; // variable permettant d'accÃ©der aux donnÃ©es du modÃ¨le

    int _scoreView;

    //HORLOGES
    sf::Clock _clock;
    sf::Clock _clockNuke;
```



```
    sf::Clock _clockCopter;
    sf::Clock _clockMovingElem;
    sf::Clock _clockStaticElem;
    bool _doubleJumpBonusActivated = false;
    bool _gameOver = false;
    bool _colStaticElement = false;
    bool _colMovingElement = false;
    bool _colTree = false;
    bool _colHelico = false;
    int _col_x, _col_y, _posXCopter, _posYCopter, _posXElem, _posYElem;
    int _nuke_x, _nuke_y;
    bool _left = false;
    bool _nukeActivation = false;
    bool _jump = false;
    bool _invincible = false;
    sf::RectangleShape _flashBomb;
    int _opacity = 0;
    bool _endNukeFlash = false;

    sf::RectangleShape _lifeLevel;
    sf::RectangleShape _delimitation{sf::Vector2f{1200,5}};

    // ELEMENTS GRAPHIQUES OBSTACLES
    GraphicElement* _tree;
    GraphicElement* _rock;
    // ELEMENTS GRAPHIQUES ANIMES OBSTACLES
    AnimatedGraphicElement* _helico;
    AnimatedGraphicElement* _bird;
    AnimatedGraphicElement* _wolf;
    AnimatedGraphicElement* _eagle;

    // ELEMENTS GRAPHIQUES BONUS
    GraphicElement* _bonusInvincible;
    GraphicElement* _bonusLife;
    GraphicElement* _bonusDoubleJump;
    GraphicElement* _bonusDoublePoint;
    GraphicElement* _bonusNuke;

    // ELEMENTS GRAPHIQUES ANIMES PIECES
    AnimatedGraphicElement* _coin;

    // ELEMENT GRAPHIQUE LIFEBAR
    GraphicElement* _lifebar;

    // ELEMENT GRAPHIQUE YETI JUMP
    GraphicElement* _yetiJump;
    GraphicElement* _yetiJump2;
    // ELEMENT GRAPHIQUE ANIME YETI & AVATAR
    AnimatedGraphicElement* _yeti;
    AnimatedGraphicElement* _yetiFace;

    // ELEMENT GRAPHIQUE ANIME SHIELD
    AnimatedGraphicElement* _shield;

    // ELEMENT GRAPHIQUE ANIME COLLISION
    AnimatedGraphicElement* _nuke;
    AnimatedGraphicElement* _impact;
    AnimatedGraphicElement* _explosion;

    // VECTEURS DE LECTURE DES FEUILLES DE SPRITES
    // PIECES
    std::vector<sf::IntRect> AnimCoin;
    // SHIELD
    std::vector<sf::IntRect> AnimShield;
    std::vector<sf::IntRect> AnimShield2;
    // OBSTACLES
    std::vector<sf::IntRect> AnimWolf;
    std::vector<sf::IntRect> AnimEagle;
    std::vector<sf::IntRect> AnimHelico;
```

```
    std::vector<sf::IntRect> AnimBird;
    // COLLISION
    std::vector<sf::IntRect> AnimNuke;
    std::vector<sf::IntRect> AnimImpact;
    std::vector<sf::IntRect> AnimExplosion;
    // YETI
    std::vector<sf::IntRect> AnimYetiForward;
    std::vector<sf::IntRect> AnimYetiBackward;
    std::vector<sf::IntRect> AnimYetiStand;
    std::vector<sf::IntRect> AnimYetiHit;

    // TEXTE DES INFOS DU JEU
    sf::Text _score_distance;
    sf::Text _lifeInfo;
    sf::Text _textPause;
    sf::Text _takenCoins;
    sf::Text _overText;
    sf::Text _highScores;

    // SONS DU JEU
    sf::Sound _soundCoins;
    sf::Sound _soundBonus;
    sf::Sound _soundHealthBonus;
    sf::Sound _soundWolf;
    sf::Sound _soundEagle;
    sf::Sound _soundNuke;
    sf::Sound _soundHelico;
    sf::Sound _soundImpact;

    // BUFFER DE CHARGEMENT DES SONS
    sf::SoundBuffer _bufferCoin;
    sf::SoundBuffer _bufferBonus;
    sf::SoundBuffer _bufferHealthBonus;
    sf::SoundBuffer _bufferWolf;
    sf::SoundBuffer _bufferEagle;
    sf::SoundBuffer _bufferImpact;
    sf::SoundBuffer _bufferHelico;
    sf::SoundBuffer _bufferNuke;

    // TEXTURE DES ELEMENTS GRAPHIQUES
    sf::Texture _TextureNuke;
    sf::Texture _TextureTree;
    sf::Texture _TextureRock;
    sf::Texture _TextureLifebar;
    sf::Texture _TextureBonusNuke;
    sf::Texture _TextureBonusInvincible;
    sf::Texture _TextureBonusLife;
    sf::Texture _TextureBonusPoint;
    sf::Texture _TextureBonusJump;
    sf::Texture _TextureSilverCoin;
    sf::Texture _TextureGoldCoin;
    sf::Texture _TextureShield;
    sf::Texture _TextureShield2;
    sf::Texture _TextureWolf;
    sf::Texture _TextureEagle;
    sf::Texture _TextureCollision;
    sf::Texture _TextureYeti;
    sf::Texture _TextureHelico;
    sf::Texture _TextureBird;
    sf::Texture _TextureExplosion;
    sf::Texture _TextureYetiJump;
    sf::Texture _TextureYetiJump2;
    sf::Texture _TextureYetiFaces;

    // STRING VARIABLES CONTENANT DIVERSES INFORMATIONS
    std::string _stringTextPause;
    std::string _stringResume;
    std::string _stringRules;
    std::string _stringBacktoMenu;
```



```
    std::string _stringHighScores;

    /*** PARTIE PAUSE ***/

    // BOOLEENS DE JEU
    bool _pause = false;
    bool _game = true;

    // BOUTONS DU MENU PAUSE
    Button* _resume_game;
    Button* _rules;
    Button* _backToMenu;

    // TITRE DE LA PAUSE
    sf::Text _pauseTitle;

    // RECTANGLE DU HEADER DE LA PAUSE
    sf::RectangleShape _header;

public:
    GameView(int w, int h, sf::RenderWindow *window);
    GameView() = delete;
    ~GameView() override;
    void draw() override;
    void synchronize() override;
    bool treatEvents() override;
    void setModel(Model * model);
    void cuttingSprite();
    bool loadImages();
    void cleanSprite();
    void loadSound();
    bool getPause();
    bool checkScore();
};

#endif // GAMEVIEW_H
```

```
#include "GraphicElement.h"




//=========================================================================
============
// Description : Constructeur de GraphicElement
// Auteur : Guillaume Nedelec,  Nicolas Marcilloux
// Date : 25/02/16
// InterÃªt : permet de creer des GraphicELement Ã  partir d'une texture et de q
uatre entiers
//=========================================================================
============
GraphicElement::GraphicElement(sf::Texture &image, int x, int y, int w, int h)
{
    this->setTexture(image);
    this->setPosition(sf::Vector2f(x,y));
    this->_w = w;
    this->_h = h;
}




//=================================================
// Description : Destructeur de GraphicElement
// Auteur : Guillaume Nedelec
// Date : 25/02/16
// InterÃªt : permet de detruire l'objet
//=================================================
GraphicElement::~GraphicElement() {}




//=================================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 25/02/16
// InterÃªt : permet de dessiner sur une fenÃªtre, un Ã©lÃ©ment graphique
//=================================================================
void GraphicElement::draw(sf::RenderWindow * window)
{
   window->draw(*this);
}




//=========================================================================
=
// Description : Modification de la taille
// Auteur : Guillaume Nedelec,  Nicolas Marcilloux
// Date : 25/02/16
// InterÃªt : permet de modifier la largeur et la longueur des Ã©lÃ©ments graphi
ques
//=========================================================================
=
void GraphicElement::resize(int w, int h)
{
    float widht = w;
    float height = h;
    sf::FloatRect bb = this->getLocalBounds();
    float width_factor = widht / bb.width;
    float height_factor = height / bb.height;
```



```
    this->setScale(width_factor, height_factor);
}




//=================================================================
// Description : Accesseur de largeur
// Auteur :  Nicolas Marcilloux
// Date : 22/05/16
// InterÃªt : permet de d'obtenir la largeur de l'Ã©lÃ©ment graphique
//=================================================================
int GraphicElement::get_w() const
{
    return _w;
}




//=================================================================
// Description : Accesseur de longueur
// Auteur : Nicolas Marcilloux
// Date : 22/05/16
// InterÃªt : permet de d'obtenir la longueur de l'Ã©lÃ©ment graphique
//=================================================================
int GraphicElement::get_h() const
{
    return _h;
}
```

```
#ifndef GRAPHICELEMENT_H
#define GRAPHICELEMENT_H

#include <SFML/Graphics.hpp>

//============================================================================
// Description:
//
// Cette classe permet de gÃ©rer tous les sprites (non animÃ©s) du jeu et de pou
voir changer leur taille
//============================================================================
class GraphicElement:public sf::Sprite
{
protected:
    int _w; //Largeur de l'image
    int _h; //Hauteur de l'image
public:
    GraphicElement(sf::Texture &image, int x, int y, int w, int h);
    virtual ~GraphicElement();
    virtual void draw(sf::RenderWindow * window);
    virtual void resize(int w, int h);
    int get_w() const;
    int get_h() const;
};

#endif // GRAPHICELEMENT_H
```

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




//=========================================================================
// Description : Constructeur d'Intro
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de creer des Intro Ã  partir d'une fenetre et de deux entie
rs
//=========================================================================
Intro::Intro(int w, int h, sf::RenderWindow *window) :
    View{w,h,window}
{
    if (!_logoTexture.loadFromFile(LOGO_BDX_IUT))
        std::cerr << "ERROR when loading image file: " << LOGO_BDX_IUT << std::e
ndl;
    if (!_logoIUTTexture.loadFromFile(LOGO_IUT))
        std::cerr << "ERROR when loading image file: " << LOGO_IUT << std::endl;
    if (!_logoBdxTexture.loadFromFile(LOGO_BDX))
        std::cerr << "ERROR when loading image file: " << LOGO_BDX << std::endl;

    _cpt = 0;

    _opacity = sf::RectangleShape{sf::Vector2f{(float)(_w),(float)(_h)}};
    _opacity.setFillColor(sf::Color(0,0,0,255));

    _title.setFont(_font);
    _title.setString("Runner Project");
    _title.setColor(sf::Color::Black);
    _title.setPosition(405,25);
    _title.setCharacterSize(58);

    _date.setFont(_font);
    _date.setString("2015 - 2016");
    _date.setColor(sf::Color::Black);
    _date.setPosition(540,100);
    _date.setCharacterSize(24);

    _logo = new GraphicElement(_logoTexture, 0.f, 0.f, 100,100);
    _logo->setPosition(125,200);

    _logo_ent = new GraphicElement(_logoBdxTexture, 0.f, 0.f, 100,100);
    _logo_ent->setPosition(480,250);

    _logo_iut = new GraphicElement(_logoIUTTexture, 0.f, 0.f, 100,100);
    _logo_iut->setPosition(875,210);

    _developersInfosPart1.setFont(_font);
    _developersInfosPart1.setString("developed by");
    _developersInfosPart1.setColor(sf::Color::Black);
    _developersInfosPart1.setPosition(515,500);
    _developersInfosPart1.setCharacterSize(24);

    _developersInfosPart2.setFont(_font);
    _developersInfosPart2.setString("Guillaume Nedelec & Nicolas Marcilloux");
    _developersInfosPart2.setColor(sf::Color::Black);
    _developersInfosPart2.setPosition(275,530);
    _developersInfosPart2.setCharacterSize(36);
}
```



```
//=========================================================================
=========================================
// Description : Destructeur d'Intro
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
uement dans l'Intro et de supprimer cette derniÃ¨re
//=========================================================================
=========================================
Intro::~Intro()
{
    delete _logo;
    delete _logo_iut;
    delete _logo_ent;
}




//=========================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dessiner sur une fenÃªtre, une Intro
//=========================================================
void Intro::draw()
{
    _window->clear(sf::Color::White);
    _window->draw(_title);
    _window->draw(_date);
    _logo->draw(_window);
    _logo_iut->draw(_window);
    _logo_ent->draw(_window);
    _window->draw(_developersInfosPart1);
    _window->draw(_developersInfosPart2);
    _window->draw(_opacity);
    _window->display();
}




//=============================================
// Description : Synchronisation de l'Ã©cran
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de creer un de fondu en Intro
//=============================================
void Intro::synchronize()
{
    if(!_endAnimation)
    {
        if (_cpt < 255 && !_nextOpacityStep)
        {
            _opacity.setFillColor(sf::Color(0,0,0,255-_cpt));
            _cpt+=2;
            _clock.restart();
        }
        else if(_cpt>= 255 && !_nextOpacityStep)
        {
            _cpt=255;
            sf::Time time = _clock.getElapsedTime();
            if(time.asSeconds() > 2.50)
            {
                _nextOpacityStep = true;
            }
```

```
        }
        else if(_nextOpacityStep)
        {
            _opacity.setFillColor(sf::Color(0,0,0,255-_cpt));
            _cpt-=2;
            if(_cpt <= 0)
            {
                _endAnimation = true;
                _result = false;
            }
        }
    }
}




//========================================================================
========================
// Description : Detecteur et traitement d'Ã©venements
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les Ã©venements de l'utilisateur et de les t
raiter en fonction de ceux-ci
//========================================================================
========================
bool Intro::treatEvents()
{
    bool result = false;
    if(_window->isOpen())
    {
        result = _result;
        sf::Event event;

        while (_window->pollEvent(event))
        {
            if(event.type == sf::Event::Closed)
            {
                _window->close();
                result = false;
            }

            else if(event.type == sf::Event::MouseButtonPressed || event.type ==
 sf::Event::KeyPressed)
            {
                _nextOpacityStep = true;
                _window->setMouseCursorVisible(false);
            }
        }
    }
    return result;
}
```

```
#ifndef INTRO_H
#define INTRO_H

#include "View.h"

//============================================================================
// Description:
//
// Cette classe permet l'affichage d'un ecran d'introduction au lancement de l'a
pplication
// Cet Ã©cran montre quelques informations du cadre du projet
//============================================================================
class Intro : public View
{
private:
    const std::string LOGO_BDX_IUT = "Images/Logo/logo.png";
    const std::string LOGO_IUT = "Images/Logo/logo_iut.png";
    const std::string LOGO_BDX = "Images/Logo/logo_Bdx.png";

    sf::Text _title; //titre du projet
    sf::Text _date; //Annee de rÃ©alisation

    //Images des diffÃ©rents logos
    GraphicElement* _logo;
    GraphicElement* _logo_iut;
    GraphicElement* _logo_ent;

    //Texture des images prÃ©cÃ©dentes
    sf::Texture _logoTexture;
    sf::Texture _logoIUTTexture;
    sf::Texture _logoBdxTexture;

    //Informations sur les developpeurs
    sf::Text _developersInfosPart1;
    sf::Text _developersInfosPart2;

    //Permet de rÃ©alisÃ© un fondu
    sf::RectangleShape _opacity;

    //Serie de variable nÃ©cÃ©saire au bon dÃ©roulement du fondu
    int _cpt;
    bool _nextOpacityStep = false;
    bool _endAnimation = false;
    bool _result = true;
    sf::Clock _clock;

public:
    Intro(int w, int h, sf::RenderWindow * window);
    ~Intro();
    void draw() override;
    void synchronize() override;
    bool treatEvents() override;
};

#endif // INTRO_H
```

```
#ifndef SETTINGS_H
#define SETTINGS_H

#include "View.h"
#include "Button.h"
#include "Model.h"

class Model;

//=========================================================================
// Description:
//
// Cette classe est un menu permettant de gÃ©rer les options du jeu
//=========================================================================
class Settings : public View
{
private:
    Model* _model; //Modele du jeu pour appliquer les options sÃ©lÃ©ctionnÃ©es
    Button* _backToMenu; //Bouton permettant le retour au menu principal
    sf::Text _title; //Titre du sous menu
    sf::RectangleShape _titleBorder; //Cadre du titre
    sf::RectangleShape _struct; //Structure ou sont affichÃ©es les donnÃ©es

    sf::Text _difficultyTitle; //EnoncÃ© du paramÃ¨tre (DifficultÃ©)
    sf::Text _difficultyParam; //Valeur de la difficultÃ©
    sf::CircleShape _decreasementDifficulty; //Petit triangle permettant de bais
ser la difficultÃ©
    sf::CircleShape _addingDifficulty; //Idem pour l'augmenter

    sf::Text _musicInfo; //EnoncÃ© du paramÃ¨tre (Musique)
    sf::Text _musicParam; //Valeur du paramÃ¨tre
    sf::Text _SoundEffectInfo; //EnoncÃ© du paramÃ¨tre (Effets Sonores)
    sf::Text _SoundEffectParam; //Valeur du paramÃ¨tre

    sf::Text _languageInfo; //EnoncÃ© du paramÃ¨tre (Effets Sonores)
    sf::Text _languageParam; //Valeur du paramÃ¨tre
    sf::CircleShape _changeUpLanguage; //Petit triangle permettant de baisser la
 difficultÃ©
    sf::CircleShape _changeDownLanguage; //Idem pour l'augmenter

    std::string _stringTitle;
    std::string _stringDifficultyTitle;
    std::string _stringDifficultyParam;
    std::string _stringMusic;
    std::string _stringMusicParam;
    std::string _stringSoundsEffects;
    std::string _stringSoundsEffectsParam;
    std::string _stringLanguage;
    std::string _stringLanguageParam;
    std::string _stringBackMenu;

public:
    Settings(int w, int h, sf::RenderWindow *window);
    ~Settings();
    void draw() override;
    void synchronize() override;
    bool treatEvents() override;
    void setModel(Model* model);
};

#endif // SETTINGS_H
```

```
#include "SlidingBackground.h"

//=========================================================================
==
// Description : Constructeur de SlidingBackground
// Auteur : Guillaume Nedelec
// Date : 20/05/16
// InterÃªt : Permet de construire des arriÃ¨res plans dÃ©filants
//=========================================================================
==
SlidingBackground::SlidingBackground(sf::Texture &image, int w, int h, unsigned
int speed):
    _left{image,0,0,w,h},
    _right{image,w,0,w,h},
    _width{w},
    _height{h},
    _speed{speed}
{
    _left.setTexture(image);
    _right.setTexture(image);
    _left.setPosition(0,0);
    _right.setPosition(w,0);
}

//=========================================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/03/16
// InterÃªt : Permet de dessiner le fond dans la fenetre et de le faire defiler
Ã  chaque Ã©tape
//=========================================================================
void SlidingBackground::draw(sf::RenderWindow * &window)
{
    _left.setPosition(_left.getPosition().x - (_speed), _left.getPosition().y );
    _right.setPosition(_right.getPosition().x - (_speed), _right.getPosition().y
 );

    if(_right.getPosition().x < 0)
    {
        _left.setPosition(0,_left.getPosition().y);
        _right.setPosition(_width, _right.getPosition().y);
    }

    window->draw(_left);
    window->draw(_right);
}

//=========================================================================
// Description : Mutateur de speed
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/03/16
// InterÃªt : Permet de modifier la vitesse de dÃ©filement
//=========================================================================
void SlidingBackground::setSpeed(unsigned int speed)
{
    this->_speed = speed;
}
```

```
#ifndef SLIDINGBACKGROUND_H
#define SLIDINGBACKGROUND_H

#include "GraphicElement.h"

//=========================================================================
// Description:
//
// Cette classe permet de gÃ©rer les fonds dÃ©filants
//=========================================================================
class SlidingBackground {
private:
    GraphicElement _left; //Image en partant du cotÃ© gauche
    GraphicElement _right; //Image en partant de la droite (vers la droite)
    //Les images de _left et _right se succÃ¨dent pour un affichage constant.
    int _width; //Largeur de l'image
    int _height; //Hauteur de l'image
    unsigned int _speed = 0; //Vitesse de dÃ©filement
public:
    SlidingBackground(sf::Texture &image, int w, int h, unsigned int speed);
    void draw(sf::RenderWindow *&window);
    void setSpeed(unsigned int speed);
};

#endif // SLIDINGBACKGROUND_H
```

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

using namespace std;

sf::Music       View::_music;
bool            View::_charge = false;
bool            View::_musicBegin = false;
bool            View::_sounds = true;
sf::Font        View::_fontTitle;
sf::Font        View::_font;
Language        View::_lang = English;

//=========================================================================
=======
// Description : Constructeur de View
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/03/16
// InterÃªt : permet de construire une fenetre avec des Ã©lÃ©ments
//=========================================================================
=======
View::View(int w, int h, sf::RenderWindow *window): _w{w}, _h{h}, _window{window
}
{
    _fontTitle.loadFromFile("Font/frow.ttf");
    _font.loadFromFile("Font/Antique Olive.ttf");

    if (!_TextureBackground1.loadFromFile(BACKGROUND_IMAGE))
        std::cerr << "ERROR when loading image file: " << BACKGROUND_IMAGE << st
d::endl;

    if (!_TextureBackground2.loadFromFile(BACKGROUND_IMAGE2))
       std::cerr << "ERROR when loading image file: " << BACKGROUND_IMAGE2 << st
d::endl;

    if (!_TextureCursorClicPressed.loadFromFile(CURSOR_CLIC))
        std::cerr << "ERROR when loading image file: " << CURSOR_CLIC << std::en
dl;

    if (!_TextureCursorClicReleased.loadFromFile(CURSOR))
       std::cerr << "ERROR when loading image file: " << CURSOR << std::endl;

    if (!_TextureMusicButton.loadFromFile(MUSIC_BUTTON))
        std::cerr << "ERROR when loading image file: " << MUSIC_BUTTON << std::e
ndl;

    if(!_charge)
    {
        if(!_music.openFromFile(SOUND))
            std::cerr << "ERROR when loading audio file: " << SOUND << std::endl
;
        _charge = true;
        _music.setLoop(true);
        _music.setVolume(20);
     }

    _cursor = new GraphicElement(_TextureCursorClicReleased,0.f, 0.f,50,50);
    _musicButton = new GraphicElement(_TextureMusicButton,0.f,0.f, 100,100);
    _soundButton = new GraphicElement(_TextureMusicButton,0.f,0.f, 100,100);
    _background1 = new SlidingBackground(_TextureBackground1,_w, _h,3);
    _background2 = new SlidingBackground(_TextureBackground2,_w, _h,1);
}

//=========================================================================
=========================================
// Description : Destructeur de View
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/03/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
```



```
uement dans le GameView et de supprimer ce dernier
//=========================================================================
=========================================
View::~View()
{
    delete _background1;
    delete _background2;
    delete _musicButton;
    delete _cursor;
    delete _soundButton;
}

//=========================================================================
=========================================
// Description : Action de dessin
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de dessiner les arriÃ¨res plan dÃ©filants
//=========================================================================
=========================================
void View::draw()
{
    _window->clear();
    _background2->draw(_window);
    _background1->draw(_window);
}

//=========================================================================
=========================================
// Description : Mutateurs de Statut
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : Permet de trouver dans quel Ã©tat est l'utilisateur
//=========================================================================
=========================================
Statut View::getStatut()
{
    return _statut;
}

//=========================================================================
=========================================
// Description : Synchronisation
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : Permet de mettre Ã  jour l'affichage
//=========================================================================
=========================================
void View::synchronize()
{
    if(_music.getStatus() == _music.Playing)
    {
        _musicButton->setTextureRect(sf::IntRect(0,0,115,115));
    }
    else
    {
        _musicButton->setTextureRect(sf::IntRect(134,0,115,115));
    }
    if(_sounds)
    {
        _soundButton->setTextureRect(sf::IntRect(0,136,115,115));
    }
    else
    {
        _soundButton->setTextureRect(sf::IntRect(125,136,115,115));
    }
    _musicButton->resize(75,75);
    _soundButton->resize(75,75);
}
```

```
#ifndef _VIEW_
#define _VIEW_

#include <SFML/Audio.hpp>
#include "AnimatedGraphicElement.h"
#include "SlidingBackground.h"

//Indique dans quel est le statut ou l'utilisateur se trouve
enum Statut{MENU,GAME,RANKING,SETTINGS, RULES};
//Indique la langue utilisÃ©
enum Language{English, French, Spanish, German};

//========================================================================
// Description:
//
// Cette classe est la base de la vue du jeu. Elle comprend les Ã©lÃ©ments commu
ns Ã  toutes les vues utilisÃ©es
// (Menu principal, jeu, meilleurs score etc...)
//========================================================================
class View {
protected:
    const std::string MUSIC_BUTTON = "Images/Boutons/MusicButtons.png";      //I
mage pour le bouton de musique
    const std::string SOUND = "Sounds/Music/Action.wav";
//Lien de la musique
    const std::string BACKGROUND_IMAGE = "Images/Background/sapins.png";
//Image du premier fond dÃ©filant
    const std::string BACKGROUND_IMAGE2 = "Images/Background/montain.png";
//Image du second fond dÃ©filant
    const std::string CURSOR = "Images/Curseur/cursor.png";
//Image du curseur
    const std::string CURSOR_CLIC = "Images/Curseur/cursor2.png";
//Image du curseur lors d'un clic

    int _w, _h; //Largeur et hauteur de la fenetre
    sf::RenderWindow * _window; //fenetre d'affichage

    Statut _statut; //Statut du programme (Menu, jeu, ect...)

    static bool  _charge; //Indique si la musique a Ã©tÃ© chargÃ© (pour ne le fa
ire qu'une seule fois)
    static bool _musicBegin;
    static sf::Music _music; //Musique du jeu
    static bool _sounds; //Indique si es effets sonores sont activÃ© ou non

    sf::Vector2i _window_position; //Position de la fenetre dans l'Ã©cran
    sf::Vector2i _mouse_position; //Position de la souris dans la fenetre
    float _mouseX ; //Position X de la souris
    float _mouseY; //Position Y de la souris

    //Texture relative aux fonds ainsi qu'au curseur et au bouton de la musique
    sf::Texture _TextureBackground1;
    sf::Texture _TextureBackground2;
    sf::Texture _TextureCursorClicPressed;
    sf::Texture _TextureCursorClicReleased;
    sf::Texture _TextureMusicButton;

    GraphicElement* _musicButton; //Image du bouton musique
    GraphicElement* _soundButton; //Image du bouton musique
    GraphicElement* _cursor; //Element graphique du curseur

    SlidingBackground* _background1; //Fond dÃ©filant 1
    SlidingBackground* _background2; //Fond dÃ©filant 2

    static Language _lang; //Langue du jeu

public:
    static sf::Font _fontTitle; //Font utilisÃ© dans le jeu
    static sf::Font _font; //Font utilisÃ© dans le jeu
```



```
    View(int w, int h, sf::RenderWindow *window);
    virtual ~View();
    virtual void draw();
    virtual bool treatEvents() =0;
    virtual void synchronize()=0;
    Statut getStatut();
};

#endif
```

```
#include "Menu.h"

using namespace std;




//=========================================================================
// Description : Constructeur de Menu
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de creer des Menu Ã  partir d'une fenetre et de deux entier
s
//=========================================================================
Menu::Menu(int w, int h, sf::RenderWindow* window) :
    View{w,h, window}
{
    _statut = MENU;
    if(!_musicBegin)
    {
        _music.play();
        _musicBegin = true;
    }

    _musicButton->setPosition(50,500);
    _soundButton->setPosition(160,500);

    _new_game = new Button(300,75,sf::Color(18,182,209,150), _stringNewGame);
    _quit = new Button(300,75,sf::Color::Blue, _stringQuit);
    _ranking = new Button(300,75,sf::Color::Blue, _stringRanking);
    _settings = new Button(300,75,sf::Color::Blue, _stringSettings);
    _rules = new Button(300,75,sf::Color::Blue, _stringRules);
    _license = new Button(200,50,sf::Color::Blue, _stringLicense);

    _new_game->setPosition(450,125);
    _settings->setPosition(450,220);
    _ranking->setPosition(450,315);
    _quit->setPosition(450,505);
    _rules->setPosition(450,410);
    _license->setPosition(975,530);

    _buttons.insert(_new_game);
    _buttons.insert(_quit);
    _buttons.insert(_ranking);
    _buttons.insert(_settings);
    _buttons.insert(_rules);
    _buttons.insert(_license);

    _title.setFont(_fontTitle);
    _title.setColor(sf::Color::Black);
    _title.setCharacterSize(48);

    _licenseContent.setFont(_font);
    _licenseContent.setColor(sf::Color::Black);

    _header = sf::RectangleShape{sf::Vector2f{600,60}};
    _header.setOutlineColor(sf::Color::Black);
    _header.setFillColor(sf::Color(18,140,225,150));
    _header.setOutlineThickness(7);
    _header.setPosition(300, 30);

    _struct = sf::RectangleShape{sf::Vector2f{600,450}};
    _struct.setOutlineColor(sf::Color::Black);
    _struct.setFillColor(sf::Color(18,140,225,150));
    _struct.setOutlineThickness(2);
    _struct.setPosition(300, 115);
}
```



```
//=========================================================================
=======================================
// Description : Destructeur d'Intro
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
uement dans le menu et de supprimer ce dernier
//=========================================================================
=======================================
Menu::~Menu()
{
    for(auto e : _buttons)
    {
        delete e;
    }
    _buttons.clear();
}




//=====================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dessiner sur une fenÃªtre, un menu
//=====================================================
void Menu::draw()
{
    View::draw();
    _musicButton->draw(_window);
    _soundButton->draw(_window);

    if(_mainMenu)
    {
        for(auto e : _buttons)
        {
            e->draw(_window);
        }
    }
    if(_licenseMenu)
    {
        _window->draw(_struct);
        _license->draw(_window);
        _window->draw(_licenseContent);
    }

    _window->draw(_header);
    _window->draw(_title);
    _cursor->draw(_window);
    _window->display();
}




//=========================================================================
========================
// Description : Detecteur et traitement d'Ã©venements
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les Ã©venements de l'utilisateur et de les t
raiter en fonction de ceux-ci
```

```
//============================================================================
=======================
bool Menu::treatEvents()
{
    bool result = false;
    if(_window->isOpen())
    {
        result = true;
        sf::Event event;

        _window_position = (_window->getPosition());
        _mouse_position = (sf::Mouse::getPosition());
        _mouseX = ((_mouse_position.x) - (_window_position.x)) -20;
        _mouseY = ((_mouse_position.y) - (_window_position.y)) +35;

        while (_window->pollEvent(event))
        {
            if ((event.type == sf::Event::Closed) || ((event.type == sf::Event::
KeyPressed) && (event.key.code == sf::Keyboard::Escape)))
            {
                _window->close();
                result = false;
            }

            if(event.type == sf::Event::MouseButtonPressed && event.mouseButton.
button == sf::Mouse::Left)
            {
                sf::Vector2f blocPosition = _musicButton->getPosition();
                sf::Vector2f blocSoundPosition = _soundButton->getPosition();

                if (_license->getSelected())
                {
                    if(_mainMenu)
                    {
                        _mainMenu = false;
                        _licenseMenu = true;
                    }
                    else if (!_mainMenu)
                    {
                        _mainMenu = true;
                        _licenseMenu = false;
                    }
                }

                if(_mainMenu)
                {
                    if (_new_game->getSelected())
                    {
                        _statut= GAME;
                        result = false;
                    }

                    else if(_quit->getSelected())
                    {
                        _window->close();
                        result = false;
                    }
                    else if(_ranking->getSelected())
                    {
                        _statut= RANKING;
                        result = false;
                    }
                    else if(_settings->getSelected())
                    {
                        _statut= SETTINGS;
                        result = false;
                    }
                    else if(_rules->getSelected())
                    {
```



```
                        _statut = RULES;
                        result = false;
                    }
                }

                if(_mouseX >= blocPosition.x-20 &&  _mouseX <= blocPosition.x+_m
usicButton->get_w()-40
                        && _mouseY >= blocPosition.y && _mouseY <= blocPosition.
y+_musicButton->get_h()-20)
                {
                    if(_music.getStatus() == _music.Playing)
                    {
                        _music.pause();
                    }

                    else
                    {
                        _music.play();
                    }
                }
                if(_mouseX >= blocSoundPosition.x-20 &&  _mouseX <= blocSoundPos
ition.x+_musicButton->get_w()-40
                        && _mouseY >= blocSoundPosition.y && _mouseY <= blocSoun
dPosition.y+_musicButton->get_h()-20)
                {
                    if(_sounds)
                    {
                        _sounds = false;
                    }

                    else
                    {
                        _sounds = true;
                    }
                }
                _cursor->setTexture(_TextureCursorClicPressed);
            }

            else if(event.type == sf::Event::MouseButtonReleased && event.mouseB
utton.button == sf::Mouse::Left)
            {
                _cursor->setTexture(_TextureCursorClicReleased);
            }

            for(Button*b : _buttons)
            {
                if(_mouseX >= b->get_x()-17 && _mouseX <= b->get_x()+b->get_w()-
15
                        && _mouseY >= b->get_y()-7 && _mouseY <= b->get_y()+b->g
et_h())
                    b->setSelection(true);

                else
                    b->setSelection(false);
            }
        }
    }
    return result;
}




//=========================================
// Description : Synchronisation du menu
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : met Ã  jour l'affichage du menu
```

```
//=======================================
void Menu::synchronize()
{
    View::synchronize();
    for(Button* b : _buttons)
    {
        if(b->getSelected())
        {
            b->rectColorChange(sf::Color(18,225,225,180));
        }
        else
        {
            b->rectColorChange(sf::Color(18,140,225,230));
        }
    }

    _cursor->setPosition(_mouseX, _mouseY);

    for(auto b : _buttons)
    {
        b->textSize(30);
    }

    _license->textSize(30);

    if(_lang == English)
    {
        _stringNewGame = "New Game";
        _stringQuit = "Quit";
        _stringRanking = "High Scores";
        _stringSettings = "Settings";

        if(_mainMenu)
        {
            _stringTitle = "The Yeti Runner";
            _stringLicense = "License";
            _title.setPosition(380, 30);
            _license->setTextPosition(1020,535);
        }
        else
        {
            _stringTitle = "License";
            _stringLicense = "Main Menu";
            _title.setPosition(500, 30);
            _license->setTextPosition(1000,535);
        }

        _stringRules = "Rules";

        _quit->setTextPosition(565,520);
        _ranking->setTextPosition(525,330);
        _new_game->setTextPosition(525,140);
        _settings->setTextPosition(540,235);
        _rules->setTextPosition(565,425);
    }

    else if(_lang == French)
    {
        _stringNewGame = "Nouvelle Partie";
        _stringQuit = "Quitter";
        _stringRanking = "Meilleurs Scores";
        _stringSettings = "Options";

        _license->textSize(24);
        if(_mainMenu)
        {
            _stringTitle = "Yeti Runner";
            _stringLicense = "Licence";
            _title.setPosition(425, 30);
```



```
            _license->setTextPosition(1030,540);
        }
        else
        {
            _stringTitle = "Licence";
            _stringLicense = "Menu principal";
            _title.setPosition(500, 30);
            _license->setTextPosition(995,540);
        }

        _stringRules = "Regles du Jeu";

        _quit->setTextPosition(550,520);
        _ranking->setTextPosition(485,330);
        _new_game->setTextPosition(490,140);
        _settings->setTextPosition(550,235);
        _rules->setTextPosition(505,425);

    }
    else if(_lang == Spanish)
    {
        _stringNewGame = "Nuevo Juego";
        _stringQuit = "Dejar";
        _stringRanking = "Mejores Puntuaciones";
        _stringSettings = "Opciones";
        _license->textSize(24);
        if(_mainMenu)
        {
            _stringTitle = "El Yeti Corredor";
            _stringLicense = "Licencia";
            _title.setPosition(370, 30);
            _license->setTextPosition(1030,540);
        }
        else
        {
            _stringTitle = "Licencia";
            _stringLicense = "menu principal";
            _title.setPosition(500, 30);
            _license->setTextPosition(995,540);
        }
        _stringRules = "Reglas del juego";

        _quit->setTextPosition(565,525);
        _ranking->setTextPosition(465,335);
        _new_game->setTextPosition(525,145);
        _settings->setTextPosition(545,240);
        _rules->setTextPosition(497,430);

        _quit->textSize(26);
        _ranking->textSize(26);
        _new_game->textSize(26);
        _settings->textSize(26);
        _rules->textSize(26);
    }

    else if(_lang == German)
    {
        _stringNewGame = "Neues Spiel";
        _stringQuit = "Verlassen";
        _stringRanking = "Highscores";
        _stringSettings = "Optionen";
        if(_mainMenu)
        {
            _stringTitle = "Der Yeti Runner";
            _stringLicense = "Lizenz";
            _title.setPosition(390, 30);
            _license->setTextPosition(1035,535);
        }
        else
```

```
        {
            _stringTitle = "Lizenz";
            _stringLicense = "Hauptmenu";
            _title.setPosition(500, 30);
            _license->setTextPosition(995,535);
        }
        _stringRules = "Regeln des Spiels";

        _quit->setTextPosition(525,520);
        _ranking->setTextPosition(520,330);
        _new_game->setTextPosition(515,140);
        _settings->setTextPosition(530,235);
        _rules->setTextPosition(480,425);
    }

    _content = "Emeline Vinzio \n\n"
             + string("Mageker\n\n")
             + "http://retrogamezone.co.uk/metalslug/info.htm \n\n"
             + "Game chefs \n\n"
             + "Gussprint \n\n"
             + "KitsuneRedWolf \n\n"
             + "Mndarrr \n\n"
             + "SEGA \n\n"
             + "TechoKami \n\n"
             + "vecteezy \n\n"
             + "SMT13/Yoshi101 \n\n"
             + "Spork Thug Typography";

    _licenseContent.setPosition(321, 135);
    _licenseContent.setCharacterSize(18);
    _licenseContent.setString(_content);

    _new_game->setText(_stringNewGame);
    _quit->setText(_stringQuit);
    _ranking->setText(_stringRanking);
    _settings->setText(_stringSettings);
    _rules->setText(_stringRules);
    _title.setString(_stringTitle);
    _license->setText(_stringLicense);
}
```

```
#ifndef MENU_H
#define MENU_H

#include "View.h"
#include "Button.h"

//============================================================================
// Description:
//
// Cette classe permet de crÃ©er un menu avec les fonctionnalitÃ©s nÃ©cÃ©ssaires
 Ã  son fonctionnement
// (Gestion des boutons ect...)
//============================================================================
class Menu : public View
{
private:
    std::set<Button*> _buttons; //Conteneur de tous les boutons prÃ©sents dans l
e menu.
    Button* _new_game; //Bouton permettant l'accÃ¨s Ã  la phase de jeu
    Button* _quit; //Bouton pour quitter
    Button* _settings; //Bouton pour accÃ©der aux obstacles
    Button* _ranking; //Bouton pour accÃ©der aux classements
    Button* _rules; //Bouton pour accÃ©der aux rÃ¨gles du jeu
    sf::Text _title; //Titre du jeu
    sf::RectangleShape _header; //Rectangle de prÃ©sentation au niveau du titre


    //Textes prÃ©sents sur cet Ã©cran
    std::string _stringNewGame;
    std::string _stringQuit;
    std::string _stringRanking;
    std::string _stringSettings;
    std::string _stringTitle;
    std::string _stringRules;

    Button* _license;
    std::string _stringLicense;
    sf::RectangleShape _struct;
    bool _mainMenu = true;
    bool _licenseMenu = false;
    sf::Text _licenseContent;
    std::string _content;


public:
    Menu(int w, int h, sf::RenderWindow *window);
    ~Menu();
    void draw() override;
    void synchronize() override;
    bool treatEvents() override;
};

#endif // MENU_H
```

```
#include "Model.h"
#include <fstream>
#include <iostream>

using namespace std;

//=========================================================================
=
// Description : Constructeur de modÃ¨le
// Auteur : Guillaume Nedelec
// Date : 01/02/16
// InterÃªt : permet de creer un modÃ¨le (squelette du jeu) Ã  partir de deux en
tier
//=========================================================================
=
Model::Model(int w, int h)
    :  _w(w), _h(h), _player{nullptr} {

    _player = new Player(INIT_POSX_PLAYER, INIT_POSY_PLAYER, SIZE_PLAYER, SIZE_P
LAYER , 0, 0); //Construction de la balle (le joueur)
}




//=========================================================================
=======================================
// Description : Destructeur de modÃ¨le
// Auteur : Guillaume Nedelec
// Date : 01/02/16
// InterÃªt : permet de dÃ©sallouer toutes les case mÃ©moires allouÃ©es dynamiqu
ement dans le modÃ¨le et de supprimer ce dernier
//=========================================================================
=======================================
Model::~Model(){

    //Desallocation de tous les pointeurs du modele : Obstacle, PiÃ¨ce et Bonus
ainsi que la balle.
    delete _player;
    for(Obstacle * e : _obstacles)
        delete e;
    for(Coin * e : _coins)
        delete e;
    for(Bonus * e : _bonus)
        delete e;
    for(MovableElement * e : _trash)
        delete e;

    _obstacles.clear();
    _coins.clear();
    _bonus.clear();
    _trash.clear();
}




//=========================================================================
=======================================
// Description : Gestion de l'apparition des elements mobiles
// Auteur : Guillaume Nedelec
// Date : 22/02/16
// InterÃªt : permet de gÃ©nÃ©rer les les objets mobiles (sauf le joueur) sans q
u'ils puissent apparaitre les uns sur les autres
//=========================================================================
=======================================
void Model::elementApparition()
{
    //DÃ©finie un dÃ©lai alÃ©atoire d'apparition d'un obstacle qui ne change pas
```



```
 tant qu'aucun obstacle est apparu
    if(!_obstacleAppears)
    {
        apparitionTimeObstacle = getDuration(Ennemies);
        _obstacleAppears = true; //Permet de ne plus modifier le dÃ©lai d'appari
tion des obstacles
    }

    //Idem pour les pieces
    if(!_coinAppears)
    {
        apparitionTimeCoin = getDuration(Points);
        _coinAppears = true; //Permet de ne plus modifier le dÃ©lai d'apparition
 des pieces
    }

    //Idem pour les bonus
    if(!_bonusAppears)
    {
        apparitionTimeBonus = getDuration(PowerUp);
        _bonusAppears = true; //Permet de ne plus modifier le dÃ©lai d'apparitio
n des bonus
    }

    //Apparition d'un obstacle
    if(_cptFrameObstacle%apparitionTimeObstacle == 0)
    {
        addElement(Ennemies);
        _cptFrameObstacle=0;
        _obstacleAppears = false;
    }

    //Apparition de piÃ¨ces
    if(_cptFrameCoin%apparitionTimeCoin == 0)
    {
        addElement(Points);
        _cptFrameCoin=0;
        _coinAppears = false;
    }

    //Apparition de bonus
    if(_cptFrameBonus%apparitionTimeBonus == 0)
    {
        addElement(PowerUp);
        _cptFrameBonus=0;
        _bonusAppears = false;
    }
}


//=========================================================================
// Description : Deroulement des Ã©tapes temporelles (par frames)
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 10/03/16
// InterÃªt : permet de faire avancer le jeu en fonction des frames
//=========================================================================
void Model::nextStep()
{
    elementApparition(); //Fais apparaitre les divers Ã©lÃ©ments mobiles
    elementManagement(); //GÃ¨re les differents Ã©vÃ©nements liÃ©s aux objets mo
biles
    movePlayer(); //Permet le dÃ©placement de la balle

    //Desactive les effets du bonus de doublage des points attrapÃ©s une fois le
 temps dÃ©passÃ©
    if(_cptFrameActiveBonusPoint%ACTIVE_TIME_BONUS == 0)
    {
        _cptFrameActiveBonusPoint = 1; //Remise du compteur Ã  la valeur initial
e (!= 0 pour ne pas remplir la condition du if)
```

```
        EndDoublePointBonus(); //Annule les effets du bonus
    }

    //Desactive les effets du bonus d'invincibilite attrapÃ© une fois le temps d
Ã©passÃ©
    if(_cptFrameActiveInvincibleBonus%ACTIVE_TIME_BONUS == 0)
    {
        _invincibleBonusActive = false; //Indique le fait que le bonus n'est plu
s actif
        _cptFrameActiveInvincibleBonus = 1; //Remise du compteur Ã  la valeur in
itiale (!= 0 pour ne pas remplir la condition du if)
        EndInvincibleBonus(); //Annule les effets du bonus
    }

    //Desactive les effets du bonus de double saut attrapÃ© une fois le temps dÃ
©passÃ©
    if(_cptFrameActiveDoubleJumpBonus%ACTIVE_TIME_BONUS == 0)
    {
        _doubleJumpBonusActive = false; //Indique le fait que le bonus n'est plu
s actif
        _cptFrameActiveDoubleJumpBonus = 1; //Remise du compteur Ã  la valeur in
itiale (!= 0 pour ne pas remplir la condition du if)
        EndDoubleJumpBonus(); //Annule les effets du bonus
    }

    //Accelere le jeu toutes les 2000 frames
    if(_cptFrameScore%1500 == 0)
    {
        _speedStaticElement-= 1;         //Vitesse nÃ©gative car dÃ©placement ve
rs la gauche
        _speedMovingElement -=1;

        /**********************************************************************
************************************************************/

        for(Obstacle* e : _obstacles)
        {
            e->set_dx(_speedStaticElement);
            if(e->get_height() == Ground)
            {
                if(e->get_type() == ObstacleType3)
                    e->set_dx(_speedMovingElement);
            }
            else if(e->get_height() == Up_Air)
            {
                e->set_dx(_speedMovingElement);
            }
        }
        for(Coin* e : _coins)
            e->set_dx(_speedStaticElement);
        for(Bonus* e : _bonus)
            e->set_dx(_speedStaticElement);
    }
    incrementation(); //Incremente tous les compteurs utilisÃ©s
}




//=========================================================================
============================================================
// Description : Gestion des elements mobiles
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 15/03/16
// InterÃªt : permet de gÃ©rer les Ã©lÃ©ments mobiles (hors joueur) : les dÃ©pla
cement, les collisions, les pseudo-suppression (insertion dans la 'trash')
//=========================================================================
============================================================
```



```
void Model::elementManagement()
{
    deleteElement();

    for(auto e : _obstacles) // On gÃ¨re chaque obstacle contenu dans le set d'o
bstacle
    {
        if(_player->collision(e)) //Si la balle rentre en collision avec l'obsta
cle
        {
            _player->damages(e->get_type()); // On enleve de la vie Ã  la balle
selon le type de l'obstacle
            _trash.insert(e);
        }

        else if((e->get_x() + e->get_w()) < 0) //Si l'obstacle sort de l'ecran
        {
            _trash.insert(e);
        }

        else
        {
            e->move(); //On dÃ©place l'obstacle
        }
    }

    for(auto e : _coins) // On gÃ¨re toutes les piÃ¨ces contenues dans le set de
 piÃ¨ce
    {
        for(auto f : _obstacles)
        {
            if(f->collision(e) && f->get_type() != ObstacleType3)
            {
                _trash.insert(e);
            }
        }

        for(auto f : _bonus)
        {
            if(f->collision(e))
            {
                _trash.insert(e);
            }
        }

        if(_player->collision(e)) //Si il y a une collision entre la balle et la
 piÃ¨ce
        {
            addToScore(); // On ajoute des points au score
            _trash.insert(e);
            _takenCoins++;
        }

        else if((e->get_x() + e->get_w()) < 0 ) //Si la piÃ¨ce sort de l'ecran
        {
            _trash.insert(e);
        }

        else
        {
            e->move(); //On dÃ©place la piÃ¨ce
        }
    }

    for(auto e : _bonus) //On gere tous les bonus du set de bonus
    {
        for(auto f : _obstacles)
        {
            if(f->collision(e) && f->get_type() != ObstacleType3)
```

```
            {
                _trash.insert(e);
            }
        }

        if(_player->collision(e)) //Si la balle rentre en collision avec un bonu
s
        {
            bonusEffects(e->get_type()); //On active les effets bonus selon le t
ype du bonus
            _trash.insert(e);
        }

        else if((e->get_x() + e->get_w()) < 0) //Si le bonus sort de l'ecran
        {
            _trash.insert(e);
        }

        else
        {
            e->move(); // On deplace le bonus
        }
    }
}




//=============================================================================
============================
// Description : Suppression des objets mobiles
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 23/03/16
// InterÃªt : permet de supprimer dÃ©finitivement les objets mobiles et de dÃ©sa
llouer leur case mÃ©moire associÃ©e
//=============================================================================
============================
void Model::deleteElement()
{
    for(auto e : _trash)
    {
        if(dynamic_cast<Obstacle*>(e) != nullptr)
            _obstacles.erase(_obstacles.find(dynamic_cast<Obstacle*>(e)));

        else if(dynamic_cast<Coin*>(e) != nullptr)
            _coins.erase(_coins.find(dynamic_cast<Coin*>(e)));

        else if(dynamic_cast<Bonus*>(e) != nullptr)
            _bonus.erase(_bonus.find(dynamic_cast<Bonus*>(e)));

        delete e;
    }
    _trash.clear();
}




//=============================================================================
===========
// Description : incrementation des compteurs de frames
// Auteur : Guillaume Nedelec
// Date : 10/03/16
// InterÃªt : permet d'incrementer les compteurs gÃ©rant les les Ã©tapes tempore
lles du modÃ¨le
//=============================================================================
===========
```



```
void Model::incrementation()
{
    _cptFrameScore++; //Compteur permettant de calculer le score
    _cptFrameBonus++; //Compteur permettant de gerer l'apparition des bonus
    _cptFrameCoin++; //Compteur permettant de gerer l'apparition des piÃ¨ces
    _cptFrameObstacle++; //Compteur permettant de gerer l'apparition des obstacl
es

    if(_doublePointBonusActive) //Si le bonus de point est activÃ©
        _cptFrameActiveBonusPoint++; //Compteur permettant de calculer le temps
d'activation du bonus

    if(_invincibleBonusActive) //Si le bonus d'invincibilite est activÃ©
        _cptFrameActiveInvincibleBonus++; //Compteur permettant de calculer le t
emps d'activation du bonus

    if(_doubleJumpBonusActive) //Si le bonus de double saut est activÃ©
        _cptFrameActiveDoubleJumpBonus++; //Compteur permettant de calculer le t
emps d'activation du bonus

}




//=============================================================================
// Description : Accesseur de Position du joueur
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les positions actuelles x et y du joueur
//=============================================================================
void Model::getPlayerPosition(int &x, int &y) const
{
    x = _player->get_x(); //Affecte les coordonnÃ©es x de la balle
    y = _player->get_y(); //Affecte les coordonnÃ©es y de la balle
}




//=====================================================
// Description : Mouvement du joueur
// Auteur : Guillaume Nedelec
// Date : 11/02/16
// InterÃªt : permet de gerer les dÃ©placements du joueur
//=====================================================
void Model::movePlayer()
{
    int x, y;
    this->getPlayerPosition(x,y); //Recuperation des coordonnÃ©es de la balle da
ns x et y

    if(_player->getLeft()&& x>0+25) //Si la balle n'est pas en dehors de l'ecran
 (Ã  gauche)
        _player->set_dx(_speedStaticElement-2); //On affecte un dÃ©placement hor
izontal de -6 (vers la gauche)

    else if(_player->getRight() && x<(_w-25)) //Si la balle n'est pas en dehors
de l'ecran (Ã  droite)
        _player->set_dx(-(_speedStaticElement)+2); //On affecte un dÃ©placement
horizontal de 6 (vers la droite)

    else
        _player->set_dx(0); //La position de la balle ne change pas sur l'axe de
s abscisses
```

```
    _player->move(); //Appel du dÃ©placement de la balle
}




//================================================
// Description : Accesseur du joueur
// Auteur : Nicolas Marcilloux
// Date : 20/2/16
// InterÃªt : permet d'acceder aux mÃ©thodes du joueur
//================================================
Player* Model::getPlayer() const
{
    return _player;
}




//================================================================================
================
// Description : Accesseur de la position d'un obstacle
// Auteur : Nicolas Marcilloux
// Date : 12/02/16
// InterÃªt : permet de rÃ©cupÃ©rer la position d'un obstacle et de la sauvegard
er dans deux entiers
//================================================================================
================

void Model::getObstaclePosition(Obstacle *obst, int &x, int &y) const
{
    x = obst->get_x();
    y = obst->get_y();
}




//================================================================================
================
// Description : Accesseur de la position d'une piÃ¨ce
// Auteur : Nicolas Marcilloux
// Date : 12/02/16
// InterÃªt : permet de rÃ©cupÃ©rer la position d'une piÃ¨ce et de la sauvegarde
r dans deux entiers
//================================================================================
================
void Model::getCoinPosition(Coin *coin, int &x, int &y) const
{
    x = coin->get_x();
    y = coin->get_y();
}




//================================================================================
================
// Description : Accesseur de la position d'un bonus
// Auteur : Nicolas Marcilloux
```



```
// Date : 12/02/16
// InterÃªt : permet de rÃ©cupÃ©rer la position d'un bonus et de la sauvegarder
dans deux entiers
//================================================================================
================
void Model::getBonusPosition(Bonus *bonus, int &x, int &y) const
{
    x = bonus->get_x();
    y = bonus->get_y();
}




//================================================================
// Description : Incrementation du score
// Auteur : Guillaume Nedelec
// Date : 16/02/16
// InterÃªt : permet d'incrÃ©menter le score en fonction des frames
//================================================================
void Model::scoreCalculation()
{
    //Toute les 35 frames (environ 1 seconde) le score s'incremente de 1
    if(_cptFrameScore%35==0)
    {
        _score++;
    }
}




//================================================
// Description : Accesseur du score
// Auteur : Nicolas Marcilloux
// Date : 12/02/16
// InterÃªt : permet de rÃ©cupÃ©rer la valeur du score
//================================================
unsigned int Model::getScore() const
{
    return _score;
}




//================================================================================
// Description : Accesseur des piÃ¨ces rÃ©cupÃ©rÃ©es
// Auteur : Nicolas Marcilloux
// Date : 15/05/16
// InterÃªt : permet de rÃ©cupÃ©rer le nombre piÃ¨ces rÃ©cupÃ©rÃ©es par le joueu
r
//================================================================================
unsigned int Model::getTakenCoins() const
{
    return _takenCoins;
}




//================================================================
// Description : Ajout d'Ã©lements mobiles
// Auteur : Nicolas Marcilloux
// Date : 14/03/16
```

```
// InterÃªt : permet de construire les Ã©lÃ©ments mobiles de faÃ§on alÃ©atoire
//============================================================================
void Model::addElement(Element elem)
{
    int elementType;
    int elementHeight;
    //---------------------------------OBSTACLES--------------------------------
--//
    if (elem==Ennemies)
    {
        elementType = rand()%3; //Permet une gÃ©nÃ©ration alÃ©atoire du type de
l'Ã©lÃ©ment
        elementHeight = rand()%6; //Permet une gÃ©nÃ©ration alÃ©atoire de la Hau
teur (au sol ou en l'air)

        if((elementHeight <=4 )) // Obstacles "au sol" (5 chances sur 9)
        {
            if(elementType == ObstacleType1) //ajout d'un obstacle de type 0 dan
s le set
            {
                Obstacle* obst = new Obstacle(INIT_POSX_MOVABLE_ELEMENT, INIT_PO
SY_GROUND, WIDTH_ROCK, HEIGHT_ROCK, _speedStaticElement, 0, ObstacleType1, Groun
d); // LITTLE ROCK
                _obstacles.insert(obst);
            }

            else if(elementType == ObstacleType2)
            {
                Obstacle* obst = new Obstacle(INIT_POSX_MOVABLE_ELEMENT, INIT_PO
SY_TREE+10, WIDTH_TREE, HEIGHT_TREE, _speedStaticElement, 0, ObstacleType2, Grou
nd); // TREE
                _obstacles.insert(obst);
            }

            else if(elementType == ObstacleType3)
            {
                Obstacle* obst = new Obstacle(INIT_POSX_MOVABLE_ELEMENT, 425, WI
DTH_WOLF, HEIGHT_WOLF, _speedMovingElement, 0, ObstacleType3, Ground); // WOLF
                _obstacles.insert(obst);
            }
        }

        else if(elementHeight > 4) // Obstacles "en l'air" (4 chances sur 9)
        {
            if(elementType == ObstacleType1)
            {
                Obstacle* obst = new Obstacle(INIT_POSX_MOVABLE_ELEMENT, INIT_PO
SY_UP_AIR, WIDTH_BIRD, HEIGHT_BIRD, _speedMovingElement, 0, ObstacleType1, Up_Ai
r); // MISSILE
                _obstacles.insert(obst);
            }

            if(elementType == ObstacleType2)
            {
                Obstacle* obst = new Obstacle(INIT_POSX_MOVABLE_ELEMENT, INIT_PO
SY_UP_AIR, WIDTH_EAGLE, HEIGHT_EAGLE, _speedMovingElement, 0, ObstacleType2, Up_
Air); // EAGLE
                _obstacles.insert(obst);
            }

            else if(elementType == ObstacleType3)
            {
                Obstacle* obst = new Obstacle(INIT_POSX_MOVABLE_ELEMENT, INIT_PO
SY_UP_AIR-50, WIDTH_HELICO, HEIGHT_HELICO, _speedMovingElement, 0, ObstacleType3
, Up_Air); // HELICO
                _obstacles.insert(obst);
            }
        }
    }
```



```
    //---------------------------------COINS---------------------------------//

    else if (elem==Points)
    {
        elementType = rand()%3;
        elementHeight = rand()%8;

        if(elementHeight <= 4) // PiÃ¨ces "au sol" ( 5 chances sur 8)
        {
            if(elementType == SeriesOf3) // serie de 3 piÃ¨ces d'affilÃ©e
            {
                Coin* coin = new Coin(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GROUN
D, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin2 = new Coin(INIT_POSX_MOVABLE_ELEMENT+50, INIT_POSY_G
ROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin3 = new Coin(INIT_POSX_MOVABLE_ELEMENT+100, INIT_POSY_
GROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                _coins.insert(coin);
                _coins.insert(coin2);
                _coins.insert(coin3);
            }

            else if(elementType == SeriesOf4) // serie de 4 piÃ¨ces d'affilÃ©e
            {
                Coin* coin = new Coin(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GROUN
D, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin2 = new Coin(INIT_POSX_MOVABLE_ELEMENT+50, INIT_POSY_G
ROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin3 = new Coin(INIT_POSX_MOVABLE_ELEMENT+100, INIT_POSY_
GROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin4 = new Coin(INIT_POSX_MOVABLE_ELEMENT+150, INIT_POSY_
GROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                _coins.insert(coin);
                _coins.insert(coin2);
                _coins.insert(coin3);
                _coins.insert(coin4);
            }

            else if(elementType == SeriesOf5) // serie de 5 piÃ¨ces d'affilÃ©e
            {
                Coin* coin = new Coin(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GROUN
D, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin2 = new Coin(INIT_POSX_MOVABLE_ELEMENT+50, INIT_POSY_G
ROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin3 = new Coin(INIT_POSX_MOVABLE_ELEMENT+100, INIT_POSY_
GROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin4 = new Coin(INIT_POSX_MOVABLE_ELEMENT+150, INIT_POSY_
GROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                Coin* coin5 = new Coin(INIT_POSX_MOVABLE_ELEMENT+200, INIT_POSY_
GROUND, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Ground);
                _coins.insert(coin);
                _coins.insert(coin2);
                _coins.insert(coin3);
                _coins.insert(coin4);
                _coins.insert(coin5);
            }
        }

        else if(elementHeight > 4) // PiÃ¨ces "en l'air" (3 chances sur 8)
        {
            if(elementType == SeriesOf3)
            {
                Coin* coin = new Coin(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MID_A
IR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin2 = new Coin(INIT_POSX_MOVABLE_ELEMENT+50, INIT_POSY_M
ID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin3 = new Coin(INIT_POSX_MOVABLE_ELEMENT+100, INIT_POSY_
MID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
```

```
                _coins.insert(coin);
                _coins.insert(coin2);
                _coins.insert(coin3);

            }

            else if(elementType == SeriesOf4)
            {
                Coin* coin = new Coin(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MID_A
IR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin2 = new Coin(INIT_POSX_MOVABLE_ELEMENT+50, INIT_POSY_M
ID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin3 = new Coin(INIT_POSX_MOVABLE_ELEMENT+100, INIT_POSY_
MID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin4 = new Coin(INIT_POSX_MOVABLE_ELEMENT+150, INIT_POSY_
MID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                _coins.insert(coin);
                _coins.insert(coin2);
                _coins.insert(coin3);
                _coins.insert(coin4);
            }

            else if(elementType == SeriesOf5)
            {
                Coin* coin = new Coin(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MID_A
IR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin2 = new Coin(INIT_POSX_MOVABLE_ELEMENT+50, INIT_POSY_M
ID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin3 = new Coin(INIT_POSX_MOVABLE_ELEMENT+100, INIT_POSY_
MID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin4 = new Coin(INIT_POSX_MOVABLE_ELEMENT+150, INIT_POSY_
MID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                Coin* coin5 = new Coin(INIT_POSX_MOVABLE_ELEMENT+200, INIT_POSY_
MID_AIR, SIZE_COIN,SIZE_COIN, _speedStaticElement, 0, Mid_Air);
                _coins.insert(coin);
                _coins.insert(coin2);
                _coins.insert(coin3);
                _coins.insert(coin4);
                _coins.insert(coin5);
            }
        }
    }

    //---------------------------------BONUS---------------------------------//

    else if (elem==PowerUp)
    {
        elementType = rand()%30;
        elementHeight = rand()%2;

        if((elementHeight == 0)) // Bonus "au sol" (1 chance sur 2)
        {

            if(elementType <= 6) //Bonus d'invincibilitÃ©
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GR
OUND, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Invincible, Ground);
                _bonus.insert(bonus);
            }

            else if(elementType > 6 && elementType <= 14) //Bonus de santÃ©
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GR
OUND, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Health, Ground);
                _bonus.insert(bonus);
            }

            else if(elementType > 14 && elementType <= 20) //Bonus double saut
            {
```



```
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GR
OUND, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Jump, Ground);
                _bonus.insert(bonus);
            }

            else if(elementType > 20 && elementType <= 26) //Bonus Point x2
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GR
OUND, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Pointx2, Ground);
                _bonus.insert(bonus);
            }

            else  //Bonus Nucleaire
            {

                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_GR
OUND, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Nuke, Ground);
                _bonus.insert(bonus);
            }

        }

        else if((elementHeight == 1)) // Bonus "en l'air" (1 chance sur 2)
        {
            if(elementType <= 6)
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MI
D_AIR, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Invincible, Mid_Air);
                _bonus.insert(bonus);
            }

            else if(elementType > 6 && elementType <= 14)
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MI
D_AIR, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Health, Mid_Air);
                _bonus.insert(bonus);
            }

            else if(elementType > 14 && elementType <= 20)
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MI
D_AIR, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Jump, Mid_Air);
                _bonus.insert(bonus);
            }

            else if(elementType > 20 && elementType <= 26)
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MI
D_AIR, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Pointx2, Mid_Air);
                _bonus.insert(bonus);
            }

            else
            {
                Bonus* bonus = new Bonus(INIT_POSX_MOVABLE_ELEMENT, INIT_POSY_MI
D_AIR, SIZE_BONUS, SIZE_BONUS, _speedStaticElement, 0, Nuke, Mid_Air);
                _bonus.insert(bonus);
            }
        }
    }
}




//=============================================
// Description : Accesseur du set d'obstacles
// Auteur : Guillaume Nedelec
```

```
// Date : 10/03/16
// InterÃªt : permet de rÃ©cupÃ©rer le set d'obstacles
//=============================================
std::set<Obstacle *> Model::getObstacle() const
{
    return _obstacles;
}




//=============================================
// Description : Accesseur du set de piÃ¨ces
// Auteur : Guillaume Nedelec
// Date : 10/03/16
// InterÃªt : permet de rÃ©cupÃ©rer le set de piÃ¨ces
//=============================================
std::set<Coin *> Model::getCoin() const
{
    return _coins;
}




//=============================================
// Description : Accesseur du set de Bonus
// Auteur : Guillaume Nedelec
// Date : 10/03/16
// InterÃªt : permet de rÃ©cupÃ©rer le set de Bonus
//=============================================
std::set<Bonus *> Model::getBonus() const
{
    return _bonus;
}




//=============================================================================
// Description : Accesseur des dÃ©lais d'apparitions d'Ã©lÃ©ments mobiles
// Auteur : Guillaume Nedelec
// Date : 13/03/16
// InterÃªt : permet de rÃ©cupÃ©rer les dÃ©lais d'apparitions des Ã©lÃ©ments mob
iles
//=============================================================================
unsigned int Model::getDuration(Element elem) const
{
    if (elem==Ennemies) //Pour les obstacles
    {
        return (MIN_APPARITION_OBSTACLE + 2*(int)(_speedStaticElement)) + (rand(
) % ((MAX_APPARITION_OBSTACLE + (int)(_speedStaticElement)) - MIN_APPARITION_OBS
TACLE));
    }

    else if (elem==Points) // Pour les piÃ¨ces
    {
        return MIN_APPARITION_COIN + (rand() % (MAX_APPARITION_COIN - MIN_APPARI
TION_COIN));
    }

    else if(elem==PowerUp)// Pour les Bonus
    {
        return MIN_APPARITION_BONUS+ (rand() % (MAX_APPARITION_BONUS- MIN_APPARI
TION_BONUS));
    }
```



```
    else
        return 0;
}




//=====================================
// Description : Augmente le score
// Auteur : Guillaume Nedelec
// Date : 09/03/16
// InterÃªt : permet d'augmenter le score
//=====================================
void Model::addToScore()
{
    if(!_doublePointBonusActive) //Lorsque le bonus Points x2 n'est pas actif
        _score +=ADD_TO_SCORE;
    else
        _score+=ADD_TO_SCORE_WITH_BONUS;
}




//=============================================
// Description : Effets des bonus
// Auteur : Guillaume Nedelec
// Date : 15/03/16
// InterÃªt : permet d'activer les effets des bonus
//=============================================
void Model::bonusEffects(BonusType type)
{
    if(type == Invincible) //Bonus d'invincibilitÃ©
    {
        if(!_invincibleBonusActive)
        {
            _player->setInvincibleBonus(true); //Active le bonus
            _invincibleBonusActive = true; //Indique que ce bonus est activÃ©
        }
        else
            _cptFrameActiveInvincibleBonus=1;
    }
    else if(type == Health) //Bonus santÃ©
    {
        _player->addLife(); //ajoute 20 points de vie Ã  la balle
    }
    else if(type == Jump) //Bonus double saut
    {
        if(!_doubleJumpBonusActive)
        {
            _player->setDoubleJumpBonus(true);
            _doubleJumpBonusActive = true; //Indique que ce bonus est activÃ©
        }
        else
            _cptFrameActiveDoubleJumpBonus=1;
    }
    else if(type == Pointx2) // Bonus Point x2
    {
        if(!_doublePointBonusActive)
            _doublePointBonusActive = true; //Active le bonus
        else
            _cptFrameActiveBonusPoint=1;
    }
    else if(type == Nuke)
    {
        for(auto e : _obstacles)
```

```
        {
            _trash.insert(e);
        }

        for(auto e : _coins)
        {
            _trash.insert(e);
        }

        for(auto e : _bonus)
        {
            _trash.insert(e);
        }
    }
}




//=========================================================
// Description : Fin Bonus Invincible
// Auteur : Guillaume Nedelec
// Date : 10/03/16
// InterÃªt : permet de mettre fin au bonus d'invincibilitÃ©
//=========================================================
void Model::EndInvincibleBonus()
{
    _player->setInvincibleBonus(false);
}




//=================================================================
// Description : Fin Bonus x2
// Auteur : Guillaume Nedelec
// Date : 10/03/16
// InterÃªt : permet de mettre fin au bonus multiplicateur de points
//=================================================================
void Model::EndDoublePointBonus()
{
    _doublePointBonusActive = false;
}




//=========================================================
// Description : Fin Bonus Double Saut
// Auteur : Guillaume Nedelec
// Date : 10/03/16
// InterÃªt : permet de mettre fin au bonus de double saut
//=========================================================
void Model::EndDoubleJumpBonus()
{
    _player->setDoubleJumpBonus(false);
}




//=============================================================================
// Description : Accesseur de vitesse d'Ã©lÃ©ments statiques
// Auteur : Guillaume Nedelec
```



```
// Date : 12/04/16
// InterÃªt : permet de recupÃ©rer la valeur de la vitesse d'Ã©lements statiques
//=============================================================================
float Model::getSpeedElement() const
{
    return _speedStaticElement;
}




//=============================================================================
// Description : Accesseur de vitesse d'Ã©lÃ©ments Mouvants
// Auteur : Guillaume Nedelec
// Date : 12/04/16
// InterÃªt : permet de recupÃ©rer la valeur de la vitesse d'Ã©lements Mouvants
//=============================================================================
float Model::getSpeedMovingElement() const
{
    return _speedMovingElement;
}




//=================================================================
// Description : RÃ©initialisation du jeu
// Auteur : Guillaume Nedelec
// Date : 16/05/16
// InterÃªt : permet de relancer le jeu avec ses valeurs de dÃ©part
//=================================================================
void Model::reset()
{
    _player->reset();

    if(_endGame)
        bestScoresManagement();

    _score = 0;
    _takenCoins = 0;
    _cptFrameScore=1;
    _cptFrameObstacle=0;
    _cptFrameCoin=0;
    _cptFrameBonus=0;
    _cptFrameActiveBonusPoint =1;
    _cptFrameActiveInvincibleBonus =1;

    _obstacleAppears = false;
    _coinAppears = false;
    _bonusAppears = false;

    _doublePointBonusActive = false;
    _invincibleBonusActive = false;
    _doubleJumpBonusActive = false;

    if(_difficulty == Easy)
    {
        _speedStaticElement = -3;
        _speedMovingElement = -6;
    }

    else if(_difficulty == Medium)
    {
        _speedStaticElement = -4;
        _speedMovingElement = -7;
    }
```

```
    else if(_difficulty == Hard)
    {
        _speedStaticElement = -5;
        _speedMovingElement = -8;
    }

    for(Obstacle * e : _obstacles)
        delete (e);
    for(Coin * e : _coins)
        delete (e);
    for(Bonus * e : _bonus)
        delete (e);


    _obstacles.clear();
    _coins.clear();
    _bonus.clear();
    _trash.clear();
}




//=============================================================
// Description : Accesseur de l'activation du bonus x2
// Auteur : Guillaume Nedelec
// Date : 08/03/16
// InterÃªt : permet de savoir si le bonus multiplicateur est actif
//=============================================================
bool Model::getDoublePointBonus() const
{
    return _doublePointBonusActive;
}




//=================================================================
// Description : Accesseur de la 'poubelle' Ã  Ã©lÃ©ments mobiles (set)
// Auteur : Guillaume Nedelec
// Date : 15/03/16
// InterÃªt : permet d'avoir accÃ¨s Ã  la 'poubelle' Ã  Ã©lÃ©ments mobiles
//=================================================================
set<MovableElement*> Model::getRemovable() const
{
    return _trash;
}




//=============================================================================
=======
// Description : Augmentation de la difficultÃ©
// Auteur : Guillaume Nedelec
// Date : 10/05/16
// InterÃªt : permet d'augmenter la difficultÃ© ainsi que la vitesse des Ã©lÃ©me
nts mobiles
//=============================================================================
=======
void Model::addDifficulty()
{
    if(_difficulty == Medium)
    {
        _difficulty = Hard;
        _speedStaticElement = -5;
        _speedMovingElement = -8;
```



```
    }
    else if(_difficulty == Easy)
    {
        _difficulty = Medium;
        _speedStaticElement = -4;
        _speedMovingElement = -7;
    }
}




//=============================================================================
=======
// Description : Diminuation de la difficultÃ©
// Auteur : Guillaume Nedelec
// Date : 10/05/16
// InterÃªt : permet de diminuer la difficultÃ© ainsi que la vitesse des Ã©lÃ©me
nts mobiles
//=============================================================================
=======
void Model::decreaseDifficulty()
{
    if(_difficulty == Medium)
    {
        _difficulty = Easy;
        _speedStaticElement = -3;
        _speedMovingElement = -6;
    }
    else if(_difficulty == Hard)
    {
        _difficulty = Medium;
        _speedStaticElement = -4;
        _speedMovingElement = -7;
    }
}




//=========================================================
// Description : Accesseur de la difficultÃ© du jeu
// Auteur : Guillaume Nedelec
// Date : 15/03/16
// InterÃªt : permet de savoir la difficultÃ© actuelle du jeu
//=========================================================
Difficulty Model::getDifficulty()
{
    return _difficulty;
}




//=========================================================
// Description : Accesseur de la fin de jeu
// Auteur : Guillaume Nedelec
// Date : 15/03/16
// InterÃªt : permet de savoir si le jeu est terminÃ© ou non
//=========================================================
bool Model::getEndGame()
{
    return _endGame;
}
```

```
//=================================================
// Description : Mutateur de la fin de jeu
// Auteur : Guillaume Nedelec
// Date : 15/03/16
// InterÃªt : permet d'appliquer la fin du jeu ou non
//=================================================
void Model::setEndGame(bool b)
{
    _endGame = b;
}




//=============================================================================
========================
// Description : Gestion des meilleurs scores
// Auteur : Guillaume Nedelec
// Date : 22/05/16
// InterÃªt : permet de lire le fichier score.txt et de le modifier si le joueur
 rentre dans le classement
//=============================================================================
========================
void Model::bestScoresManagement()
{
    fstream f;
    int element;
    int location = -1;
    int i=0;
    string fichier = "scores.txt";
    bool foundLocation = false;
    unsigned int saveScore[5];

    std::ifstream infile(fichier);
    if(infile.good())
    {
        f.open(fichier.c_str(), ios:: in); //Ouverture en lecture

        if(f.fail())
        {
            std::cerr << "ouverture en lecture impossible" << endl;
            exit(EXIT_FAILURE);
        }
        while(!f.eof())
        {
            f >> element;
            saveScore[i] = element;
            i++;
        }
        f.close();

        i=0;

        while(!foundLocation && i<5)
        {
            if(_score > saveScore[i] )
            {
                foundLocation = true;
                location = i;
            }
            else
                i++;
        }

        if(foundLocation)
```



```
        {
            for(int i = 5; i > location; i--)
                saveScore[i] = saveScore[i-1];

            saveScore[location] = _score;
        }
    }
    else
    {
        saveScore[0] = _score;
        for(int i = 1; i < 5; i++)
            saveScore[i] = 0;
    }

    f.open(fichier.c_str(), ios::out);
    if(f.fail())
    {
        std::cerr << "ouverture en ecriture impossible" << endl;
        exit(EXIT_FAILURE);
    }
    for(i=0;i<5 ; i++)// i<5 parce qu'il n'y a que 5 meilleurs scores
    {
        f << saveScore[i] << endl; //on copie le fichier tampon dans le fichier
meilleur score en effaÃ§ant ce qu'il y avait au prÃ©alable.
    }
    f.close();
}
```

```
#ifndef _MODEL_
#define _MODEL_

#include "Player.h"
#include "Obstacle.h"
#include "Coin.h"
#include "Bonus.h"
#include <set>

//TAILLES DES DIFFERENTS ELEMENTS MOBILES DU JEU
static const unsigned int SIZE_BONUS = 50;
static const unsigned int SIZE_COIN = 30;

static const unsigned int WIDTH_BIRD = 75;
static const unsigned int HEIGHT_BIRD = 40;

static const unsigned int WIDTH_HELICO = 100;
static const unsigned int HEIGHT_HELICO = 100;

static const unsigned int WIDTH_EAGLE = 100;
static const unsigned int HEIGHT_EAGLE = 50;

static const unsigned int WIDTH_ROCK = 50;
static const unsigned int HEIGHT_ROCK = 50;

static const unsigned int WIDTH_TREE = 60;
static const unsigned int HEIGHT_TREE = 110;

static const unsigned int WIDTH_WOLF = 100;
static const unsigned int HEIGHT_WOLF = 75;

//Designe le type d'Ã©lÃ©ments mobiles
enum Element
{
    Ennemies,
    Points,
    PowerUp
};

//DifficultÃ©
enum Difficulty {
    Easy,
    Medium,
    Hard
};

class Player;


//========================================================================
// Description:
//
// Cette classe est le coeur du jeu. Chaque Ã©tape du jeu se dÃ©roule dans cette
 classe (dÃ©placement, apparition d'Ã©lÃ©ments ect...)
//========================================================================
class Model {
private:

    //Temps minimum et maximum d'apparition
    static const unsigned int MIN_APPARITION_OBSTACLE = 50;
    static const unsigned int MAX_APPARITION_OBSTACLE = 100;
    static const unsigned int MIN_APPARITION_COIN = 300;
    static const unsigned int MAX_APPARITION_COIN = 500;
    static const unsigned int MIN_APPARITION_BONUS = 200;
    static const unsigned int MAX_APPARITION_BONUS = 300;
    static const unsigned int ACTIVE_TIME_BONUS = 450;

    //POSITION A LA CREATION DES OBJETS
    static const int INIT_POSX_PLAYER = 100;
```



```
    static const int INIT_POSY_PLAYER = 470;
    static const int INIT_POSX_MOVABLE_ELEMENT = 1300;
    static const int INIT_POSY_GROUND = 450;
    static const int INIT_POSY_MID_AIR = 340;
    static const int INIT_POSY_UP_AIR = 280;
    static const int INIT_POSY_TREE = INIT_POSY_GROUND - HEIGHT_TREE + 25;

    //VALEUR D'AJOUT DE SCORE
    static const unsigned int ADD_TO_SCORE = 100;
    static const unsigned int ADD_TO_SCORE_WITH_BONUS = 200;

    int _w, _h; // Taille du modele
    unsigned int _score = 0;
    unsigned int _takenCoins = 0; //Nombres de piÃ¨ces rÃ©cupÃ©rÃ©es
    float _speedStaticElement = -4; //Vitesse des Ã©lÃ©ments statiques (pierre,
arbre ect..)
    float _speedMovingElement = -7; //Vitesse des Ã©lÃ©ments mobiles (hÃ©lico, a
igle ect...)
    Difficulty _difficulty = Medium;
    bool _endGame =false;

    //Compeur de frame utilisÃ© pour diverses fonctions
    unsigned int _cptFrameScore = 1;
    unsigned int _cptFrameObstacle = 0;
    unsigned int _cptFrameCoin = 0;
    unsigned int _cptFrameBonus = 0;
    unsigned int _cptFrameActiveBonusPoint = 1;
    unsigned int _cptFrameActiveInvincibleBonus = 1;
    unsigned int _cptFrameActiveDoubleJumpBonus = 1;

    //DELAIS D4APPARITION
    unsigned int apparitionTimeObstacle;
    unsigned int apparitionTimeCoin;
    unsigned int apparitionTimeBonus;

    //INDICATEUR D'APPARITION
    bool _obstacleAppears = false;
    bool _coinAppears = false;
    bool _bonusAppears = false;

    //INDICATEUR DE BONUS
    bool _doublePointBonusActive = false;
    bool _invincibleBonusActive= false;
    bool _doubleJumpBonusActive = false;

    //CONTENEURS D'ELEMENTS
    std::set<Obstacle *> _obstacles;
    std::set<Coin *> _coins;
    std::set<Bonus *> _bonus;
    std::set<MovableElement*>_trash;

public:
    Player * _player;
    Model(int w, int h);
    ~Model();
    Player* getPlayer() const;

    void nextStep();
    void getPlayerPosition(int &x, int &y) const;
    void getObstaclePosition(Obstacle *obst, int &x, int &y) const;
    void getCoinPosition(Coin *coin, int &x, int &y) const;
    void getBonusPosition(Bonus *bonus, int &x, int &y) const;
    void movePlayer();
    void scoreCalculation();
    unsigned int getScore() const;
    unsigned int getTakenCoins() const;

    std::set<Obstacle *> getObstacle() const;
    std::set<Coin *> getCoin() const;
```

```
    std::set<Bonus *> getBonus() const;
    std::set<MovableElement *> getRemovable() const;

    void addElement(Element elem);
    unsigned int getDuration(Element elem) const;

    void incrementation();
    void elementApparition();
    void elementManagement();

    void deleteElement();

    void addToScore();
    void bonusEffects(BonusType type);
    void EndInvincibleBonus();
    void EndDoublePointBonus();
    void EndDoubleJumpBonus();
    bool getDoublePointBonus() const;
    float getSpeedElement() const;
    float getSpeedMovingElement() const;
    void addDifficulty();
    void decreaseDifficulty();
    Difficulty getDifficulty();
    void setEndGame(bool b);
    bool getEndGame();
    void reset();
    void bestScoresManagement();
};

#endif
```

```
#include "MovableElement.h"




//=========================================================================
=============
// Description : Constructeur de MovableElement
// Auteur : Guillaume Nedelec,  Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de creer des MovableELement Ã  partir de quatre float et de
 2 unsigned int
//=========================================================================
=============
MovableElement::MovableElement(float x, float y, unsigned int w, unsigned int h,
 float dx, float dy)
{
    this->_x = x;
    this->_y = y;
    this->_w = w;
    this->_h = h;
    this->_dx= dx;
    this->_dy= dy;
}




//=================================================
// Description : Destructeur de MovableElement
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de detruire l'objet
//=================================================
MovableElement::~MovableElement(){}




//=========================================================================
// Description : Accesseur de position horizontale
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de d'obtenir la position horizontale de l'Ã©lÃ©ment mobile
//=========================================================================
float MovableElement::get_x() const
{
    return _x; //CoordonnÃ©es des abscisses
}




//=========================================================================
// Description : Accesseur de position verticale
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de d'obtenir la position verticale de l'Ã©lÃ©ment mobile
//=========================================================================
float MovableElement::get_y() const
{
    return _y; //CoordonnÃ©es des ordonnÃ©es
}
```



```



//=========================================================
// Description : Accesseur de largeur
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de d'obtenir la largeur de l'Ã©lÃ©ment mobile
//=========================================================
unsigned int MovableElement::get_w() const
{
    return _w; //Largeur de l'Ã©lÃ©ment
}




//=========================================================
// Description : Accesseur de longueur
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de d'obtenir la longueur de l'Ã©lÃ©ment mobile
//=========================================================
unsigned int MovableElement::get_h() const
{
    return _h; //Hauteur de l'Ã©lÃ©ment
}




//=========================================================================
// Description : Accesseur de vitesse horizontale
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de d'obtenir la vitesse horizontale de l'Ã©lÃ©ment mobile
//=========================================================================
float MovableElement::get_dx() const
{
    return _dx; //Vitesse de dÃ©placement horizontal
}




//=========================================================================
// Description : Accesseur de vitesse verticale
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de d'obtenir la vitesse verticale de l'Ã©lÃ©ment mobile
//=========================================================================
float MovableElement::get_dy() const
{
    return _dy; //Vitesse de dÃ©placement vertical
}




//=========================================================================
// Description : Mutateur de vitesse horizontale
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de modifier la vitesse horizontale de l'Ã©lÃ©ment mobile
```

```
//=====================================================================
void MovableElement::set_dx(float dx)
{
    this->_dx = dx;
}




//=====================================================================
// Description : Mutateur de vitesse verticale
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de modifier la vitesse verticale de l'Ã©lÃ©ment mobile
//=====================================================================
void MovableElement::set_dy(float dy)
{
    this->_dy = dy;
}




//=============================================================
// Description : Mouvement des Ã©lÃ©ments mobiles
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de gerer les dÃ©placements des Ã©lÃ©ments mobiles
//=============================================================
void MovableElement::move()
{
    this->_x=_x+_dx;
    this->_y=_y+_dy;
}




//=============================================================================
// Description : Mouvement des Ã©lÃ©ments mobiles
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 11/02/16
// InterÃªt : permet de dÃ©tÃ©cter si il y a collision entre deux Ã©lÃ©ment mobi
les
//=============================================================================
bool MovableElement::collision(MovableElement *m)
{
    const float leftA = _x; //Cote gauche de la balle
    const float rightA = _x + _w; //Le droit
    const float topA = _y; //Le Haut
    const float bottomA = _y + _h; //Le bas

    //Idem pour l'element mobile
    const float leftB = m->get_x();
    const float rightB = m->get_x() + m->get_w();
    const float topB = m->get_y();
    const float bottomB = m->get_y() + m->get_h();

    if(bottomA <= topB)
        return false;
    if(topA >= bottomB)
        return false;
    if(rightA <= leftB)
        return false;
    if(leftA >= rightB)
        return false;
```



```
    return true;
}
```

```
#ifndef MOVABLEELEMENT_H
#define MOVABLEELEMENT_H

//Indique la hauteur d'apparition des Ã©lÃ©ments
enum HeightElement
{
    Ground,
    Mid_Air,
    Up_Air
};

//=========================================================================
// Description:
//
// Cette classe permet la gestion des Ã©lÃ©ments mobiles du jeu (personnage, obs
tacle, bonus et piÃ¨ces)
//=========================================================================
class MovableElement
{
protected:
    float _x; //Position x
    float _y; // Position y
    unsigned int _w; //Largeur de l'objet
    unsigned int _h; //Hauteur de l'objet
    float _dy; //Vitesse verticale
    float _dx; //Vitesse horizontale
    HeightElement _height; //Indicateur de la hauteur d'apparition de l'objet

public:
    MovableElement() {}
    MovableElement(float x, float y, unsigned int w, unsigned int h, float dx, f
loat dy);
    virtual void move();
    void set_dx(float dx);
    void set_dy(float dy);
    float get_x() const;
    float get_y() const;
    unsigned int get_w() const;
    unsigned int get_h() const;
    float get_dx() const;
    float get_dy() const;
    virtual ~MovableElement();
    virtual bool collision(MovableElement *m);
};

#endif // MOVABLEELEMENT_H
```

```
#include "Obstacle.h"




//=====================================================================
==================================
// Description : Constructeur d'Obstacle
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de creer des Obstacles Ã  partir de 4 float, 2 unsigned int
, un TypeObstacles et un HeightElement
//=====================================================================
==================================
Obstacle::Obstacle(float x, float y, unsigned int w, unsigned int h, float dx, f
loat dy, TypeObstacle type, HeightElement height):
    MovableElement{x,y,w,h,dx,dy}, _type{type}
{
    _height = height;
}






//====================================
// Description : Destructeur d'Obstacle
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de detruire l'objet
//====================================
Obstacle::~Obstacle() {}






//=================================================
// Description : Accesseur du type
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de rÃ©cupÃ©rer le type de l'Obstacle
//=================================================
TypeObstacle Obstacle::get_type() const
{
    return _type;
}






//=================================================
// Description : Accesseur de la hauteur
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet de rÃ©cupÃ©rer la hauteur de l'Obstacle
//=================================================
HeightElement Obstacle::get_height() const
{
    return _height;
}
```

```
#ifndef OBSTACLE_H
#define OBSTACLE_H

#include "MovableElement.h"

//Indique le type de l'obstacle
enum TypeObstacle
{
    ObstacleType1,
    ObstacleType2,
    ObstacleType3
};

//=========================================================================
// Description:
//
// Cette classe permet de gÃ©rer les propriÃ©tÃ©s des obstacle.
//=========================================================================
class Obstacle : public MovableElement
{
private:
    TypeObstacle _type; //type de l'obstacle

public:
    Obstacle(float x, float y, unsigned int w, unsigned int h, float dx, float d
y, TypeObstacle type, HeightElement height);
    TypeObstacle get_type() const;
    HeightElement get_height() const;
    virtual ~Obstacle() override;
};

#endif // OBSTACLE_H
```

```
#include "Player.h"

using namespace std;

//=========================================================================================
=====================================
// Description : Constructeur du joueur
// Auteur : Nicolas Marcilloux et Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet de creer des un joueur comme un MovableElement
//=========================================================================================
=====================================
Player::Player(float x, float y, unsigned int w, unsigned int h, float dx, float
 dy)
    :MovableElement{x,y,w,h,dx,dy}
{}


//=========================================================================================
=====================================
// Description : Destructeur de Player
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet de dÃ©truire les objets de type Player
//=========================================================================================
=====================================
Player::~Player()
{}


//=========================================================================================
=====================================
// Description : Accesseur de left
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet d'acceder au parametre _left
//=========================================================================================
=====================================
bool Player::getLeft() const
{
    return _left; //Indique si la balle se dirige Ã  gauche
}


//=========================================================================================
=====================================
// Description : Accesseur de right
// Auteur : Nicolas Marcilloux
// Date : 10/02/16
// InterÃªt : permet d'acceder au parametre _right
//=========================================================================================
=====================================
bool Player::getRight() const
{
    return _right; //Indique si la balle se dirige Ã  droite
}


//=========================================================================================
=====================================
// Description : Mutateur de left
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet de modifier le parametre _left
//=========================================================================================
=====================================
void Player::setLeft(bool b)
{
    _left = b;
```



```
}


//=========================================================================================
=====================================
// Description : Mutateur de right
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet de modifier le parametre _right
//=========================================================================================
=====================================
void Player::setRight(bool b)
{
    _right = b;
}


//=========================================================================================
=====================================
// Description : Accesseur de jump
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet d'acceder au parametre _jump (savoir si le joueur est en sa
ut)
//=========================================================================================
=====================================
bool Player::getJump() const
{
    return _jump; //Indique si une phase de saut est activÃ©e
}

//=========================================================================================
=====================================
// Description : Mutateur de jump
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/02/16
// InterÃªt : permet de modifier le parametre _jump
//=========================================================================================
=====================================
void Player::setJump(bool b)
{
    _jump=b;
}


//=========================================================================================
=====================================
// Description : Accesseur de doubleJump
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 10/03/16
// InterÃªt : permet d'acceder au parametre _doubleJump (savoir si le joueur est
 en double saut)
//=========================================================================================
=====================================
void Player::setDoubleJump(bool b)
{
    _doubleJump = b;
}



//=========================================================================================
=====================================
// Description : Deplacement de l'objet
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 18/02/16
// InterÃªt : Permet de faire deplacer le joueur
//=========================================================================================
=====================================
```

```
void Player::move()
{

    if(_jump && !_doubleJump) //Si le saut est activÃ©
    {
        jump();
        _x+=_dx/1.5; //Deplacement horizontal limitÃ©
    }

    else if(_doubleJump) //Si le double saut est activÃ©
    {
        doubleJump();
        _x+=_dx/1.5; //Deplacement horizontal limitÃ©
    }

    else
    {
        this->_x=_x+_dx; //DÃ©placement Normal de la balle
        this->_y=_y-_dy;
    }
}


//=========================================================================================
==================================
// Description : Saut de l'objet
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 18/02/16
// InterÃªt : Permet de faire sauter le joueur
//=========================================================================================
==================================
void Player::jump()
{
    _endJump = false;

    if(_begining)
    {
        _dy = 27.8;
        _begining = false;
    }

    if(_y > MAX_HEIGHT_JUMP && !_falling)
    {
        _dy *= GRAVITE;
        _y -= _dy;
    }

   if(_y <= MAX_HEIGHT_JUMP)
    {
        _falling = true;
    }

    if(_falling && !_endJump)
    {
        _dy /= GRAVITE;
        _y += _dy;

        if(_y >= (GROUND_HEIGHT))
        {
            _dy = 0;
            _y = (GROUND_HEIGHT);
            _falling = false;
            _endJump = true;
            _begining = true;
        }
    }

    if (_endJump)
    {
```



```
        _jump = false;
    }
}


//=========================================================================================
==================================
// Description : DoubleSaut de l'objet
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 18/02/16
// InterÃªt : Permet de faire sauter le joueur alors qu'il est dÃ©ja en saut
//=========================================================================================
==================================
void Player::doubleJump()
{
    _endDoubleJump = false;

    if(_doubleJumpBegining)
    {
        _dy = 27.8;
        _doubleJumpBegining = false;
    }

    if(_y > _save_PosY - HEIGHT_JUMP && !_doubleJumpFalling)
    {
        _dy *= GRAVITE; //Ralentissement du joueur
        _y -= _dy;
    }

   if(_y <= _save_PosY - HEIGHT_JUMP)
    {
        _doubleJumpFalling = true;
    }

    if(_doubleJumpFalling && !_endDoubleJump)
    {
        _dy = (_dy/GRAVITE); //Acceleration du joueur
        _y += _dy;

        if(_y >= (GROUND_HEIGHT))
        {
            _dy = 0;
            _y = (GROUND_HEIGHT);
            _falling = false;
            _endJump = true;
            _begining = true;
            _doubleJumpFalling = false;
            _endDoubleJump = true;
            _doubleJumpBegining = true;
        }
    }

    if (_endDoubleJump)
    {
         _jump = false;
         _doubleJump = false;
    }
}


//=========================================================================================
==================================
// Description : Collision avec le joueur
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/02/16
// InterÃªt : Permet de detecter les collisions avec le joueur
//=========================================================================================
==================================
bool Player::collision(MovableElement *m)
```

```
{
    const float leftA = _x-25; //Cote gauche de la balle
    const float rightA = _x + _w-25; //Le droit
    const float topA = _y-25; //Le Haut
    const float bottomA = _y + _h-25; //Le bas

    //Idem pour l'element mobile
    const float leftB = m->get_x();
    const float rightB = m->get_x() + m->get_w();
    const float topB = m->get_y();
    const float bottomB = m->get_y() + m->get_h();

    if(bottomA <= topB)
        return false;
    if(topA >= bottomB)
        return false;
    if(rightA <= leftB)
        return false;
    if(leftA >= rightB)
        return false;

    return true;
}



//=============================================================================
==================================
// Description : Degats
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/02/16
// InterÃªt : Permet de de baisser la vie du joueur
//=============================================================================
==================================
void Player::damages(int obstacleType)
{
    if(!_invincibleBonus) //Si le bonus d'invicibilitÃ© n'est pas actif
    {
        if(obstacleType == 0)
            _life -= HEALTH_1;
        else if(obstacleType == 1)
            _life -= HEALTH_2;
        else if(obstacleType == 2)
            _life -= HEALTH_3;
    }
}


//=============================================================================
==================================
// Description : Accesseur de life
// Auteur : Nicolas Marcilloux, Guillaume NÃ©dÃ©lec
// Date : 18/02/16
// InterÃªt : Permet de recupÃ©rer le niveau de vie du joueur
//=============================================================================
==================================
int Player::getLife() const
{
    return _life;
}


//=============================================================================
==================================
// Description : Mutateur de _invincibleBonus
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Permet d'attribuer le bonus invincibilitÃ© (ou de le retirer)
//=============================================================================
```



```
==================================
void Player::setInvincibleBonus(bool b)
{
    _invincibleBonus = b;
}

//=============================================================================
==================================
// Description : Mutateur de _doubleJumpBonus
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Permet d'attribuer le bonus double saut (ou de le retirer)
//=============================================================================
==================================
void Player::setDoubleJumpBonus(bool b)
{
    _doubleJumpBonus = b;
}

//=============================================================================
==================================
// Description : Ajout de vie
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Permet de redonner de la vie au joueur (effet d'un bonus)
//=============================================================================
==================================
void Player::addLife()
{
    if(_life >= 80 ) //Pour que la vie ne depasse jamais 100
        _life = FULL_LIFE;
    else
        _life +=HEALTH_2;
}

//=============================================================================
==================================
// Description : Reinitialisation
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Reinitialise la balle comme si le jeu recommenÃ§ais
//=============================================================================
==================================
void Player::reset()
{
    _x = 100;
    _y = GROUND_HEIGHT;
    _w = SIZE_PLAYER;
    _h = SIZE_PLAYER;
    _dx= 0;
    _dy = 0;
    _left = false;
    _right = false;
    _jump = false;
    _life =FULL_LIFE;
    _invincibleBonus = false;
    _doubleJumpBonus = false;
}

//=============================================================================
==================================
// Description : Mutateur de _invincibleBonus
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Permet de savoir si le bonus double d'invincibilitÃ© est activÃ© o
u non
//=============================================================================
==================================
bool Player::getInvincibleBonus()
```

```
{
    return _invincibleBonus;
}

//=========================================================================
==================================
// Description : Accesseur de _doubleJumpBonus
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Permet de savoir si le bonus double saut est activÃ© ou non
//=========================================================================
==================================
bool Player::getDoubleJumpBonus()
{
    return _doubleJumpBonus;
}

//=========================================================================
==================================
// Description : SavePosition
// Auteur :  Guillaume NÃ©dÃ©lec
// Date : 18/03/16
// InterÃªt : Permet de sauvegarder la position y de la balle
//=========================================================================
==================================
void Player::set_y(float y)
{
    _save_PosY = y;
}
```

```
#ifndef BALL_H
#define BALL_H

#include "MovableElement.h"
#include "View.h"


static const unsigned int SIZE_PLAYER = 75;

//=========================================================================
// Description:
//
// Cette classe permet de crÃ©er le joueur et de gÃ©rer les actions dont il est
acteur (saut, collision ect...)
//=========================================================================
class Player:public MovableElement
{
private:
    static const unsigned int HEALTH_1 = 10; //Degats d'un obstacle de type 1
    static const unsigned int HEALTH_2 = 15; //Degats d'un obstacle de type 2
    static const unsigned int HEALTH_3 = 20; //Degats d'un obstacle de type 3
    static const unsigned int FULL_LIFE = 100; //Niveau de vie Maximum
    static const unsigned int MAX_HEIGHT_JUMP = 250; //CoordonnÃ©es de la hauteu
r maximum du saut
    const float GRAVITE = 0.9; //Constante gravitationnelle (pour le saut)
    static const unsigned int HEIGHT_JUMP = 220; //Hauteur d'un saut
    static const unsigned int GROUND_HEIGHT = 470; //CoordonnÃ©es de la hauteur
du sol dans le jeu

    bool _left = false; //Indique que le joueur va vers la gauche
    bool _right = false; //Indique que le joueur va vers la droite
    bool _jump = false; //Indique si le joueur est en saut
    bool _falling = false; //Indique si lae joueur est en train de tomber (dans
le saut)
    bool _begining = true; //Debut du saut
    bool _endJump = false; //Fin du saut
    bool _doubleJump = false; //Indique si le joueur est en double saut
    bool _doubleJumpFalling = false; //Indique si le joueur est en train de tomb
er (dans le  deuxiÃ¨me saut)
    bool _doubleJumpBegining = true; //Debut du deuxiÃ¨me saut
    bool _endDoubleJump = false; //Fin du duexiÃ¨me saut
    bool _invincibleBonus = false; //Bonus d'invincibilitÃ© actif ou non
    bool _doubleJumpBonus = false; //Bonus de double saut actif ou non
    int _life = FULL_LIFE; //Vie du joueur
    float _save_PosY; //Sauvegarde de la position du joueur pour l'execution du
double saut

public:
    Player() {}
    Player(float x, float y, unsigned int w, unsigned int h, float dx, float dy)
;
    ~Player();
    bool getLeft() const;
    bool getRight() const;
    bool getJump() const;
    int getLife() const;
    void setLeft(bool b);
    void setRight(bool b);
    void setJump(bool b);
    void setDoubleJump(bool b);
    void jump();
    void doubleJump();
    void move() override;

    bool collision(MovableElement *m) override;
    void damages(int obstacleType);
    void setInvincibleBonus(bool b);
    bool getInvincibleBonus();
    void setDoubleJumpBonus(bool b);
```



```
    bool getDoubleJumpBonus();
    void addLife();
    void reset();
    void set_y(float y);
};

#endif // BALL_H
```

```
#include "Ranking.h"
#include <iostream>
#include <fstream>

using namespace std;

//================================================================================
// Description : Constructeur de Ranking
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de creer un Ã©cran de score Ã  partir d'une fenetre et de d
eux entiers
//================================================================================
Ranking::Ranking(int w, int h, sf::RenderWindow *window) :
    View{w,h,window}
{
    _statut = RANKING;

    _musicButton->setPosition(50,410);
    _soundButton->setPosition(50,500);

    _backToMenu = new Button(200,75,sf::Color(18,140,225,230), _stringBackMenu);
    _backToMenu->setPosition(10,10);
    _backToMenu->setTextPosition(32,30);

    _titleBorder = sf::RectangleShape{sf::Vector2f{600,60}};
    _titleBorder.setOutlineColor(sf::Color::Black);
    _titleBorder.setFillColor(sf::Color(18,140,225,150));
    _titleBorder.setOutlineThickness(7);
    _titleBorder.setPosition(300, 30);

    _struct = sf::RectangleShape{sf::Vector2f{800,400}};
    _struct.setOutlineColor(sf::Color::Black);
    _struct.setFillColor(sf::Color(18,140,225,150));
    _struct.setOutlineThickness(2);
    _struct.setPosition(200, 130);

    _title.setFont(_fontTitle);
    _title.setColor(sf::Color::Black);
    _title.setCharacterSize(48);

    _score1.setFont(_font);
    _score1.setColor(sf::Color::White);
    _score1.setPosition(500,180);
    _score1.setCharacterSize(48);

    _score2.setFont(_font);
    _score2.setColor(sf::Color::White);
    _score2.setPosition(500,240);
    _score2.setCharacterSize(48);

    _score3.setFont(_font);
    _score3.setColor(sf::Color::White);
    _score3.setPosition(500,300);
    _score3.setCharacterSize(48);

    _score4.setFont(_font);
    _score4.setColor(sf::Color::White);
    _score4.setPosition(500,360);
    _score4.setCharacterSize(48);

    _score5.setFont(_font);
    _score5.setColor(sf::Color::White);
    _score5.setPosition(500,420);
    _score5.setCharacterSize(48);

    ReadingScores();
}
```



```
//================================================================================
========================================
// Description : Destructeur de Ranking
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
uement dans Ranking et de supprimer cette derniÃ¨re
//================================================================================
========================================
Ranking::~Ranking()
{
    delete _backToMenu;
}

//========================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dessiner sur une fenÃªtre, les Ã©lÃ©ments pour faire l'e
cran de score
//========================================================
void Ranking::draw()
{
    View::draw();
    _musicButton->draw(_window);
    _soundButton->draw(_window);
    _backToMenu->draw(_window);
    _window->draw(_titleBorder);
    _window->draw(_title);
    _window->draw(_struct);
    _window->draw(_score1);
    _window->draw(_score2);
    _window->draw(_score3);
    _window->draw(_score4);
    _window->draw(_score5);
    _cursor->draw(_window);
    _window->display();
}

//================================================================================
========================
// Description : Detecteur et traitement d'Ã©venements
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les Ã©venements de l'utilisateur et de les t
raiter en fonction de ceux-ci
//================================================================================
========================
bool Ranking::treatEvents()
{
    bool result = false;
    if(_window->isOpen())
    {
        result = true;
        sf::Event event;

        _window_position = (_window->getPosition());
        _mouse_position = (sf::Mouse::getPosition());
        _mouseX = ((_mouse_position.x) - (_window_position.x)) -20;
        _mouseY = ((_mouse_position.y) - (_window_position.y)) +35;

        while (_window->pollEvent(event))
        {
            if(event.type == sf::Event::Closed)
            {
                _window->close();
                result = false;
            }
```

```
            if(event.type == sf::Event::MouseButtonPressed && event.mouseButton.
button == sf::Mouse::Left)
            {
                sf::Vector2f blocPosition = _musicButton->getPosition();
                sf::Vector2f blocSoundPosition = _soundButton->getPosition();

                if(_backToMenu->getSelected())
                {
                    result = false;
                    _statut = MENU;
                }
                else if(_mouseX >= blocPosition.x-20 &&  _mouseX <= blocPosition
.x+_musicButton->get_w()-40
                        && _mouseY >= blocPosition.y && _mouseY <= blocPosition.
y+_musicButton->get_h()-20)
                {
                    if(_music.getStatus() == _music.Playing)
                    {
                        _music.pause();
                    }

                    else
                    {
                        _music.play();
                    }
                }
                else if(_mouseX >= blocSoundPosition.x-20 &&  _mouseX <= blocSou
ndPosition.x+_soundButton->get_w()-40
                        && _mouseY >= blocSoundPosition.y && _mouseY <= blocSoun
dPosition.y+_soundButton->get_h()-20)
                {
                    if(_sounds)
                    {
                        _sounds = false;
                    }

                    else
                    {
                        _sounds = true;
                    }
                }
                _cursor->setTexture(_TextureCursorClicPressed);
            }
            else if(event.type == sf::Event::MouseButtonReleased && event.mouseB
utton.button == sf::Mouse::Left)
            {
                _cursor->setTexture(_TextureCursorClicReleased);
            }


            if(_mouseX >= -12 && _mouseX <= 10 +_backToMenu->get_w()-15
                    && _mouseY >= 10 && _mouseY <= 10 + _backToMenu->get_h())
                _backToMenu->setSelection(true);

            else
                _backToMenu->setSelection(false);
        }
    }
    return result;
}

//=============================================
// Description : Synchronisation de l'Ã©cran
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de creer mettre Ã  jour l'affichage
//=============================================
void Ranking::synchronize()
```



```
{
    View::synchronize();

    if(_backToMenu->getSelected())
        _backToMenu->rectColorChange(sf::Color(18,225,225,180));
    else
        _backToMenu->rectColorChange(sf::Color(18,140,225,230));

    _backToMenu->textSize(24);
    _backToMenu->setTextPosition(32,30);
    if(_lang == English)
    {
        _stringBackMenu = "Back to Menu";
        _stringTitle = "Highscores";
        _title.setPosition(480,28);
    }
    else if(_lang == French)
    {
        _stringBackMenu = "Retour Menu";
        _stringTitle = "Meilleurs Scores";
        _title.setPosition(425,28);
    }

    else if(_lang == Spanish)
    {
        _stringBackMenu = "Volver al Menu";
        _stringTitle = "Mejores Puntuaciones";
        _title.setPosition(325,28);
    }

    else if(_lang == German)
    {
        _stringBackMenu = "Zuruck zum Menu";
        _stringTitle = "Highscores";
        _title.setPosition(475,28);
        _backToMenu->setTextPosition(25,30);
        _backToMenu->textSize(20);
    }

    _title.setString(_stringTitle);
    _backToMenu->setText(_stringBackMenu);

    _cursor->setPosition(_mouseX, _mouseY);
}

//=============================================
// Description : Lecture de fichier
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : Permet de lire le fichier scores pour garder les variables
//=============================================
void Ranking::ReadingScores()
{
    fstream f;
    int element;
    string fichier = "scores.txt";
    int saveScore[5];
    int i = 0;

    std::ifstream infile(fichier);

    if(infile.good())
    {
        f.open(fichier.c_str(), ios:: in); //Ouverture en lecture

        if(f.fail())
        {
            std::cerr << "ouverture en lecture impossible" << endl;
            exit(EXIT_FAILURE);
```

```
        }
        while(!f.eof())
        {
            f >> element;
            saveScore[i] = element;
            i++;
        }
        f.close();
    }
    else
    {
        for(i=0;i<5 ; i++)// i<5 parce qu'il n'y a que 5 meilleurs scores
        {
            saveScore[i] = 0; //on copie le fichier tampon dans le fichier meill
eur score en effaÃ§ant ce qu'il y avait au prÃ©alable.
        }
    }

    _score1.setString("1. " + to_string(saveScore[0]));
    _score2.setString("2. " + to_string(saveScore[1]));
    _score3.setString("3. " + to_string(saveScore[2]));
    _score4.setString("4. " + to_string(saveScore[3]));
    _score5.setString("5. " + to_string(saveScore[4]));
}
```

```
#ifndef RANKING_H
#define RANKING_H

#include "View.h"
#include "Button.h"

//============================================================================
// Description:
//
// Cette classe permet de gÃ©rer les Ã©lÃ©ments visuel du menu des meilleurs sco
res
//============================================================================
class Ranking : public View
{
private:
    Button* _backToMenu; //Bouton permettant de retourner au menu principal
    sf::Text _title; //Titre du menu des meilleurs scores
    sf::RectangleShape _titleBorder; //Cadre autour du titre
    sf::RectangleShape _struct; //Cadre ou les donnÃ©es seront affichÃ©es
    sf::Text _score1; //Meilleur score
    sf::Text _score2; //2e Meilleur score
    sf::Text _score3; //3e Meilleur score
    sf::Text _score4; //...
    sf::Text _score5;

    std::string _stringBackMenu;
    std::string _stringTitle;

public:
    Ranking(int w, int h, sf::RenderWindow *window);
    ~Ranking();
    void draw() override;
    void synchronize() override;
    bool treatEvents() override;
    void ReadingScores();
};

#endif // RANKING_H
```

```
#include "Rules.h"
#include "GameView.h"
#include <iostream>
#include <string>

using namespace std;

//============================================================================
// Description : Constructeur de Rules
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de creer un ecran de RÃ¨gles du jeu
//============================================================================
Rules::Rules(int w, int h, sf::RenderWindow *window) :
    View{w,h,window}
{
    if(loadImages())
    {
        _statut = RULES;

        _musicButton->setPosition(12,400);
        _soundButton->setPosition(12,490);

        _header = sf::RectangleShape{sf::Vector2f{600,60}};
        _header.setOutlineColor(sf::Color::Black);
        _header.setFillColor(sf::Color(18,140,225,150));
        _header.setOutlineThickness(7);
        _header.setPosition(300, 30);

        _title.setFont(_fontTitle);
        _title.setColor(sf::Color::Black);
        _title.setCharacterSize(48);

        _subTitle.setFont(_font);
        _subTitle.setColor(sf::Color::Black);
        _subTitle.setCharacterSize(32);
        _subTitle.setStyle(sf::Text::Bold | sf::Text::Italic);

        _subSubTitle.setFont(_font);
        _subSubTitle.setColor(sf::Color::Black);
        _subSubTitle.setCharacterSize(22);
        _subSubTitle.setStyle(sf::Text::Underlined);

        _subSubTitle2.setFont(_font);
        _subSubTitle2.setColor(sf::Color::Black);
        _subSubTitle2.setCharacterSize(22);
        _subSubTitle2.setStyle(sf::Text::Underlined);

        _desc1.setFont(_font);
        _desc1.setColor(sf::Color::Black);
        _desc1.setCharacterSize(16);

        _desc2.setFont(_font);
        _desc2.setColor(sf::Color::Black);
        _desc2.setCharacterSize(16);

        _desc3.setFont(_font);
        _desc3.setColor(sf::Color::Black);
        _desc3.setCharacterSize(16);

        _desc4.setFont(_font);
        _desc4.setColor(sf::Color::Black);
        _desc4.setCharacterSize(16);

        _desc5.setFont(_font);
        _desc5.setColor(sf::Color::Black);
        _desc5.setCharacterSize(16);

        _desc6.setFont(_font);
```



```
        _desc6.setColor(sf::Color::Black);
        _desc6.setCharacterSize(16);

        _struct = sf::RectangleShape{sf::Vector2f{1000,450}};
        _struct.setOutlineColor(sf::Color::Black);
        _struct.setFillColor(sf::Color(18,140,225,150));
        _struct.setOutlineThickness(2);
        _struct.setPosition(100, 115);

        _closeInfo.setFont(_font);
        _closeInfo.setColor(sf::Color::Black);
        _closeInfo.setCharacterSize(48);

        _nextPage = new Button(200,75,sf::Color(18,140,225,230), _stringNextButt
on);
        _nextPage->setPosition(990,10);
        _previousPage = new Button(200,75,sf::Color(18,140,225,230), _stringPrev
iousButton);
        _previousPage->setPosition(10,10);

        //PAGE 1
        _yeti = new AnimatedGraphicElement(AnimYeti, _TextureYeti, 200, 240, 0,
0);
        _arrow = new GraphicElement(_TextureArrow,170,360,0,0);

        //PAGE 2
        _rock = new GraphicElement(_TextureRock,160,220,50,50);
        _tree = new GraphicElement(_TextureTree,140,310,50,50);
        _wolf = new AnimatedGraphicElement(AnimWolf,_TextureWolf,100,470,50,50);
        _bird = new AnimatedGraphicElement(AnimBird, _TextureBird,620,230,50,50)
;
        _eagle = new AnimatedGraphicElement(AnimEagle,_TextureEagle,620,350,50,5
0);
        _copter = new AnimatedGraphicElement(AnimCopter,_TextureCopter,620,450,5
0,50);

        // PAGE 3
        _bonusNuke = new GraphicElement(_TextureBonusNuke, 620,220,50,50);
        _bonusHealth  = new GraphicElement(_TextureBonusLife,160,350,50,50);
        _bonusJump = new GraphicElement(_TextureBonusJump,160,480,50,50);
        _bonusShield = new GraphicElement(_TextureBonusInvincible,160,220,50,50)
;
        _bonusPoint = new GraphicElement(_TextureBonusPoint,620,350,50,50);
        _silverCoin = new AnimatedGraphicElement(AnimCoin,_TextureSilverCoin,620
,450,50,50);
        _goldCoin = new AnimatedGraphicElement(AnimCoin,_TextureGoldCoin,620,500
,50,50);
    }
}


//============================================================================
==========================================
// Description : Destructeur de Rules
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
uement dans les Regles et de supprimer cette derniÃ¨re
//============================================================================
==========================================
Rules::~Rules()
{
    delete _nextPage;
    delete _previousPage;

    delete _yeti;
    delete _arrow;

    delete _tree;
```

```
    delete _rock;
    delete _bird;
    delete _wolf;
    delete _eagle;
    delete _copter;

    delete _bonusHealth;
    delete _bonusJump;
    delete _bonusShield;
    delete _bonusNuke;
    delete _bonusPoint;
    delete _silverCoin;
    delete _goldCoin;
}

//=========================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de dessiner sur une fenÃªtre, l'ecran de regles
//=========================================================
void Rules::draw()
{
    View::draw();
    _musicButton->draw(_window);
    _soundButton->draw(_window);
    _window->draw(_struct);
    _window->draw(_header);
    _window->draw(_title);
    _window->draw(_closeInfo);


    if(_page1)
    {
        _nextPage->draw(_window);
        _yeti->draw(_window);
        _arrow->draw(_window);
        _window->draw(_subTitle);
        _window->draw(_desc1);
        _window->draw(_desc2);

    }
    else if(_page2)
    {
        _nextPage->draw(_window);
        _previousPage->draw(_window);
        _tree->draw(_window);
        _rock->draw(_window);
        _wolf->draw(_window);
        _eagle->draw(_window);
        _bird->draw(_window);
        _copter->draw(_window);
        _window->draw(_subTitle);
        _window->draw(_subSubTitle);
        _window->draw(_subSubTitle2);
        _window->draw(_desc1);
        _window->draw(_desc2);
        _window->draw(_desc3);
        _window->draw(_desc4);
        _window->draw(_desc5);
        _window->draw(_desc6);
    }
    else if(_page3)
    {
        _previousPage->draw(_window);
        _bonusHealth->draw(_window);
        _bonusNuke->draw(_window);
        _bonusPoint->draw(_window);
        _bonusJump->draw(_window);
```



```
        _bonusShield->draw(_window);
        _silverCoin->draw(_window);
        _goldCoin->draw(_window);

        _window->draw(_subTitle);
        _window->draw(_desc1);
        _window->draw(_desc2);
        _window->draw(_desc3);
        _window->draw(_desc4);
        _window->draw(_desc5);
        _window->draw(_desc6);
    }
    _cursor->draw(_window);
    _window->display();
}

//=============================================
// Description : Synchronisation de l'Ã©cran
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de mettre Ã  jour l'affichage
//=============================================
void Rules::synchronize()
{
    View::synchronize();

    _cursor->setPosition(_mouseX, _mouseY);

    _nextPage->textSize(24);
    _previousPage->textSize(24);

    if(_lang == English)
    {
        _stringTitle = "Rules";
        _stringNextButton = "Next Page";
        _stringPreviousButton = "Previous Page";
        _stringCloseInfo = "Press 'Escape' to return to the previous menu";

        if(_page1)
        {
            _stringSubTitle = "Player";
            _stringDesc1 = "Your are Yetiti, the nice Yeti of Mt Everest who for
got,\n"
                    +string("despite all his reminders, the first day of the sal
es.\n")
                    +string("So he quickly get out of his cave to find the most\
n")
                    +string("interessant prices. In his trek, he will face a lot
 of obstacles,\n")
                    +string("such as imposing firs, famished wolves or Army's he
licopter\n")
                    +string("whom confound him with his criminal cousin : Bigfoo
t\n")
                    +string("During this, he'll have to catch a lot of coins\n")
                    +string("in order to buy his clothes.");
            _stringDesc2 = "The game is only played on keyboard :\n"
                    +string("'RIGHT ARROW' to go Forward\n")
                    +string("'LEFT ARROW' to go Backward\n")
                    +string("'UP ARROW' to Jump\n")
                    +string("'ESCAPE' to Pause");

            _subTitle.setPosition(550, 120);
            _desc1.setPosition(350, 220);
            _desc2.setPosition(350, 400);
        }

        if(_page2)
        {
            _stringSubTitle = "Ennemies";
```

```
            _stringSubSubTitle = "Ground Ennemies";
            _stringSubSubTitle2 = "Flying Ennemies";
            _stringDesc1 = "ROCK : A simple block of granite\nDAMAGE : 10%\nMOVE
MENT : NO";
            _stringDesc2 = "TREE : A beautiful fir (but not least dangerous)\nDA
MAGE : 15%\nMOVEMENT : NO";
            _stringDesc3 = "WOLF : A wild famished wolf\nDAMAGE : 20%\nMOVEMENT
: YES";
            _stringDesc4 = "LITTLE BIRD : A clumsy grey bird\nDAMAGE : 10%\nMOVE
MENT : YES";
            _stringDesc5 = "BIG BIRD : A proud and fierce bald eagle\nDAMAGE : 1
5%\nMOVEMENT : YES";
            _stringDesc6 = "HELICOPTER : Death from the Skies !\nDAMAGE : 20%\nM
OVEMENT : YES";

            _subTitle.setPosition(550, 120);
            _subSubTitle.setPosition(330, 170);
            _subSubTitle2.setPosition(800, 170);
            _desc1.setPosition(250, 220);
            _desc2.setPosition(250, 350);
            _desc3.setPosition(250, 480);
            _desc4.setPosition(750, 220);
            _desc5.setPosition(750, 350);
            _desc6.setPosition(750, 480);
        }

        else if(_page3)
        {
            _stringSubTitle = "Power-ups and Coins";
            _stringDesc1 = "SHIELD : Makes you invincible during few second.";
            _stringDesc2 = "HEALTH : Restore 15% of your HP.";
            _stringDesc3 = "WINGS : Allows you to execute a double-jump.";
            _stringDesc4 = "NUKE : KABOOM !";
            _stringDesc5 = "MULTIPLIER : Doubles the value of coins. Jackpot !";
            _stringDesc6 = "COIN AND GOLDEN COIN (x2 Bonus) : Gotta catch'em all
 !";

            _subTitle.setPosition(460, 120);
            _desc1.setPosition(230, 230);
            _desc2.setPosition(230, 360);
            _desc3.setPosition(230, 490);
            _desc4.setPosition(690, 230);
            _desc5.setPosition(690, 360);
            _desc6.setPosition(690, 490);
        }

        _title.setPosition(545,28);
        _closeInfo.setPosition(380,570);
        _nextPage->setTextPosition(1035,30);
        _previousPage->setTextPosition(30,30);
    }
    else if(_lang == French)
    {
        _stringTitle = "Regles du Jeu";
        _stringNextButton = "Page Suivante";
        _stringPreviousButton = "Page Precedente";
        _stringCloseInfo = "Appuyer sur 'Echap' pour revenir au menu precedent";

        if(_page1)
        {
            _stringSubTitle = "Joueur";
            _stringDesc1 = "Vous incarnez Yetiti, le gentil Yeti du Mont Everest
 qui oublia,\n"
                    +string("malgre tous ses pense-betes, le fameux jour d'ouver
ture des soldes.\n")
                    +string("Celui-ci du donc sortir en trombe de sa caverne afi
n de denicher\n")
                    +string("les prix les plus interessants. Dans son periple il
 devra faire face\n")
```



```
                    +string("a de nombreux obstacles, tels que les imposants sap
ins, les loups affames\n")
                    +string("ou encore les helicopteres de l'armee qui le confon
dent avec son\n")
                    +string("cousin criminel : 'Bigfoot'. Au passage il devra ra
masser un maximum\n")
                    +string("de pieces pour realiser ses depenses vestimentaires
.");
            _stringDesc2 = "Le jeu ne se joue qu'au clavier' :\n"
                    +string("'FLECHE DROITE' pour avancer\n")
                    +string("'FLECHE GAUCHE' pour reculer\n")
                    +string("'FLECHE HAUT' pour sauter\n")
                    +string("'ECHAP' pour faire Pause");

            _subTitle.setPosition(550, 120);
            _desc1.setPosition(350, 220);
            _desc2.setPosition(350, 400);
        }

        if(_page2)
        {
            _stringSubTitle = "Ennemis";
            _stringSubSubTitle = "Ennemis Terrestres";
            _stringSubSubTitle2 = "Ennemis Aeriens";
            _stringDesc1 = "ROCHER : Un simple bloc de granit\nDEGAT : 10%\nMOUV
EMENT : NON";
            _stringDesc2 = "ARBRE : Un magnifique sapin\n       (mais pas moins
dangereux)\nDEGAT : 15%\nMOUVEMENT : NON";
            _stringDesc3 = "LOUP : Un loup sauvage affame\nDEGAT : 20%\nMOUVEMEN
T : OUI";
            _stringDesc4 = "PETIT OISEAU : Un oiseau gris et maladroit\nDEGAT :
10%\nMOUVEMENT : OUI";
            _stringDesc5 = "GRAND OISEAU : Un aigle fier et feroce\nDEGAT : 15%\
nMOUVEMENT : OUI";
            _stringDesc6 = "HELICOPTERE : La Mort vient du ciel !\nDEGAT : 20%\n
MOUVEMENT : OUI";

            _subTitle.setPosition(550, 120);
            _subSubTitle.setPosition(330, 170);
            _subSubTitle2.setPosition(800, 170);
            _desc1.setPosition(250, 220);
            _desc2.setPosition(250, 340);
            _desc3.setPosition(250, 480);
            _desc4.setPosition(750, 220);
            _desc5.setPosition(750, 350);
            _desc6.setPosition(750, 480);
        }

        else if(_page3)
        {
            _stringSubTitle = "Bonus et Pieces";
            _stringDesc1 = "BOUCLIER : Vous rend invincible pendant\n        que
lques secondes.";
            _stringDesc2 = "VIE : Restaure 15% de vos HP.";
            _stringDesc3 = "AILES : Vous permez d'effectuer des double-saut.";
            _stringDesc4 = "OGIVE NUCLEAIRE : KABOOM !";
            _stringDesc5 = "MULTIPLICATEUR : Double la valeur des pieces. Jackpo
t !";
            _stringDesc6 = "PIECE ET PIECE DOREE (Bonus x2) : Attrapez-les toute
s !";

            _subTitle.setPosition(480, 120);
            _desc1.setPosition(230, 230);
            _desc2.setPosition(230, 360);
            _desc3.setPosition(230, 490);
            _desc4.setPosition(690, 230);
            _desc5.setPosition(690, 360);
            _desc6.setPosition(690, 490);
        }
```

```
        _title.setPosition(460,28);
        _closeInfo.setPosition(350,570);
        _nextPage->setTextPosition(1010,30);
        _previousPage->setTextPosition(17,30);
    }
    else if(_lang == Spanish)
    {
        _stringTitle = "Reglas del Juego";
        _stringNextButton = "Pagina Siguiente";
        _stringPreviousButton = "Pagina Anterior";
        _stringCloseInfo = "Pulse 'Escape' para volver al menu anterior";

        if(_page1)
        {
            _stringSubTitle = "Jugador";
            _stringDesc1 = "Usted juega Yetiti, agradable Yeti Everest se olvido
 de que,\n"
                    +string("a pesar de todas sus think-betes, los famosos saldo
s dÃ-a de apertura.\n")
                    +string("Esto por lo tanto fuera del torbellino de su cueva
para desenterrar\n")
                    +string("el precio mas interesante. En su viaje se enfrentar
a a muchos obstaculos,\n")
                    +string("como los abetos imponentes, lobos hambrientos, los
helicÃ³pteros del ejercito\n")
                    +string("que lo confunda con su primo criminal: 'Bigfoot'. D
e paso debe recoger los\n")
                    +string("pedazos para lograr sus gastos de ropa.");
            _stringDesc2 = "El juego se juegan solo con el teclado' :\n"
                    +string("'FLECHA DERECHA' para avanzar")
                    +string("'FLECHA IZQUIERDA' para respaldar\n")
                    +string("'FLECHE UP' para saltar")
                    +string("'ESCAPE' para hacer una Pausa");

            _subTitle.setPosition(550, 120);
            _desc1.setPosition(350, 220);
            _desc2.setPosition(350, 400);
        }

        if(_page2)
        {
            _stringSubTitle = "Enemigos";
            _stringSubSubTitle = "Enemigos Terrenales";
            _stringSubSubTitle2 = "Enemigos de Aire";
            _stringDesc1 = "ROCA : Un solo bloque de granito\nDANO : 10%\nMOVIMI
ENTO : NO";
            _stringDesc2 = "ARBOL : Un abeto magnifica\n       (pero no meno pe
ligroso)\nDAMAGE : 15%\nMOVIMIENTO : NO";
            _stringDesc3 = "LOBO : Un lobo salvaje famelico\nDANO : 20%\nMOVIMIE
NTO : SI";
            _stringDesc4 = "PEQUENO PAJARO : Un ave gris y torpe\nDANO : 10%\nMO
VIMIENTO : SI";
            _stringDesc5 = "GRAN PAJARO : Un aguila orgulloso y feroz\nDANO : 15
%\nMOVIMIENTO : SI";
            _stringDesc6 = "HELICOPTERO : La Muerte viene del cielo !\nDANO : 20
%\nMOVIMIENTO : SI";

            _subTitle.setPosition(550, 120);
            _subSubTitle.setPosition(330, 170);
            _subSubTitle2.setPosition(800, 170);
            _desc1.setPosition(250, 220);
            _desc2.setPosition(250, 340);
            _desc3.setPosition(250, 480);
            _desc4.setPosition(750, 220);
            _desc5.setPosition(750, 350);
            _desc6.setPosition(750, 480);
        }
```



```
        if(_page3)
        {
            _stringSubTitle = "Bonificaciones";
            _stringDesc1 = "BLINDAR : Esto te hace invencible durante\n      alg
unos segundo.";
            _stringDesc2 = "SALUD : Restaura 15% de su HP.";
            _stringDesc3 = "ALAS : Permite realizar doble saltos";
            _stringDesc4 = "ARMA NUCLEAR : KABOOM !";
            _stringDesc5 = "MULTIPLICADOR : Duplica el valor de las monedas.\n
     Jackpot !";
            _stringDesc6 = "MONEDAS Y MONEDAS DE ORO (x2 Bonificaciones) :\n
   Atraparlos a todos !";

            _subTitle.setPosition(480, 120);
            _desc1.setPosition(230, 230);
            _desc2.setPosition(230, 360);
            _desc3.setPosition(230, 490);
            _desc4.setPosition(690, 230);
            _desc5.setPosition(690, 360);
            _desc6.setPosition(690, 490);
        }

        _title.setPosition(375,28);
        _closeInfo.setPosition(380,570);
        _nextPage->setTextPosition(1000,30);
        _previousPage->setTextPosition(20,30);


    }
    else if(_lang == German)
    {
        _stringTitle = "Spielregeln";
        _stringNextButton = "Nachste Seite";
        _stringPreviousButton = "Vorherige Seite";
        _stringCloseInfo ="DrÃ¼cken Sie 'Flucht' zurÃ¼ck zum vorherigen Menu";

        if(_page1)
        {
            _stringSubTitle = "Spieler";
            _stringDesc1 =  "Sie spielen Yetiti, nette Yeti Everest vergessen, \
n"
                    +string(" dass trotz all seiner Think-betes, die berÃ¼hmten
Salden ErÃ¶ffnungstag.\n")
                    +string("Dies deshalb aus dem Wirbelwind aus seiner HÃ¶hle d
en interessantesten\n")
                    +string("Preis ans Licht zu bringen. In seiner Reise wird er
 viele Hindernisse,\n")
                    +string(" wie zum Beispiel den hohen Tannen, hungrige WÃ¶lfe
 Gesicht, die Hubschrauber\n")
                    +string("der Armee, die ihn mit seiner kriminellen Cousin ve
rwechseln: 'Bigfoot'. Im Vorbeigehen\n")
                    +string("sollte StÃ¼cke abholen zu seiner Kleidung Kosten er
reichen.");
            _stringDesc2 = "Das Spiel wird gespielt nur mit der Tastatur' :\n"
                    +string("'PFEIL RECHT' um vorzurucken\n")
                    +string("'PFEIL LINKS' um Back\n")
                    +string("'PFEIL UP' zu springen\n")
                    +string("'FLUCHT' pausieren");

            _subTitle.setPosition(550, 120);
            _desc1.setPosition(350, 220);
            _desc2.setPosition(350, 400);
        }

        if(_page2)
        {
            _stringSubTitle = "Feinde";
            _stringSubSubTitle = "Irdischen Feinde";
            _stringSubSubTitle2 = "Air Feinde";
            _stringDesc1 = "ROCK : Ein einzelner Block aus Granit\nSCHADEN : 10%
```

```
\nBEWEGUNG : NEIN";
            _stringDesc2 = "BAUM : Ein schoner Tanne (but not least dangerous)\n
SCHADEN : 15%\nBEWEGUNG : NEIN";
            _stringDesc3 = "WOLF : Ein wilde ausgehungert Wolf\nSCHADEN : 20%\nB
EWEGUNG : JA";
            _stringDesc4 = "LITTLE BIRD : Ein umstandlich grauer Vogel\nSCHADEN
: 10%\nBEWEGUNG : JA";
            _stringDesc5 = "BIG BIRD : Ein stolzer und heftige Adler\nSCHADEN :
15%\nBEWEGUNG : JA";
            _stringDesc6 = "HELICOPTER : Der Tod kommt vom Himmel !\nSCHADEN : 2
0%\nBEWEGUNG : JA";

            _subTitle.setPosition(550, 120);
            _subSubTitle.setPosition(330, 170);
            _subSubTitle2.setPosition(800, 170);
            _desc1.setPosition(250, 220);
            _desc2.setPosition(250, 340);
            _desc3.setPosition(250, 480);
            _desc4.setPosition(750, 220);
            _desc5.setPosition(750, 350);
            _desc6.setPosition(750, 480);
        }

        if(_page3)
        {
            _stringSubTitle = "Bonus";
            _stringDesc1 = "ABSCHIRMEN : Dieser Bonus macht Sie unbesiegbar\n
    fÃ¼r einige Sekunden.";
            _stringDesc2 = "GESUNDHEIT : Erholen 15% Ihrer HP.";
            _stringDesc3 = "FLUGEL : Ermoglicht make Doppelsprunge.";
            _stringDesc4 = "KERNWAFFEN : KABOOM !";
            _stringDesc5 = "MULTIPLIKATOR : Verdopplen Sie den Wert der Munzen.\
n       Jackpot !";
            _stringDesc6 = "MUNZEN UND GOLDMUNZEN (x2 Feinde) :\n       Fangen s
ie alle !";

            _subTitle.setPosition(550, 120);
            _desc1.setPosition(230, 230);
            _desc2.setPosition(230, 360);
            _desc3.setPosition(230, 490);
            _desc4.setPosition(690, 230);
            _desc5.setPosition(690, 360);
            _desc6.setPosition(690, 490);
        }

        _title.setPosition(480,28);
        _closeInfo.setPosition(375,570);
        _nextPage->setTextPosition(1015,30);
        _previousPage->setTextPosition(20,30);
    }

    _title.setString(_stringTitle);
    _subTitle.setString(_stringSubTitle);
    _subSubTitle.setString(_stringSubSubTitle);
    _subSubTitle2.setString(_stringSubSubTitle2);
    _desc1.setString(_stringDesc1);
    _desc2.setString(_stringDesc2);
    _desc3.setString(_stringDesc3);
    _desc4.setString(_stringDesc4);
    _desc5.setString(_stringDesc5);
    _desc6.setString(_stringDesc6);
    _closeInfo.setString(_stringCloseInfo);
    _closeInfo.setCharacterSize(20);
    _nextPage->setText(_stringNextButton);
    _previousPage->setText(_stringPreviousButton);
}

//=============================================================================
=========================
```



```
// Description : Detecteur et traitement d'Ã©venements
// Auteur : Guillaume Nedelec
// Date : 23/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les Ã©venements de l'utilisateur et de les t
raiter en fonction de ceux-ci
//=============================================================================
=========================
bool Rules::treatEvents()
{
    bool result = false;
    if(_window->isOpen())
    {
        result = true;
        sf::Event event;

        _window_position = (_window->getPosition());
        _mouse_position = (sf::Mouse::getPosition());
        _mouseX = ((_mouse_position.x) - (_window_position.x)) -20;
        _mouseY = ((_mouse_position.y) - (_window_position.y)) +35;

        while (_window->pollEvent(event))
        {
            if(event.type == sf::Event::Closed)
            {
                _window->close();
                result = false;
            }

            if(event.type == sf::Event::MouseButtonPressed && event.mouseButton.
button == sf::Mouse::Left)
            {
                sf::Vector2f blocPosition = _musicButton->getPosition();
                sf::Vector2f blocSoundPosition = _soundButton->getPosition();

                if(_page1)
                {
                    if(_nextPage->getSelected())
                    {
                        _page1 = false;
                        _page2 = true;
                    }
                }
                else if(_page2)
                {
                    if(_nextPage->getSelected())
                    {
                        _page2 = false;
                        _page3 = true;
                    }
                    else if(_previousPage->getSelected())
                    {
                        _page1 = true;
                        _page2 = false;
                    }
                }
                else if(_page3)
                {
                    if(_previousPage->getSelected())
                    {
                        _page3 = false;
                        _page2 = true;
                    }
                }
                if(_mouseX >= _closeInfo.getPosition().x-20 && _mouseX <= _close
Info.getPosition().x +400
                        && _mouseY >= _closeInfo.getPosition().y -10 && _mouseY
<= _closeInfo.getPosition().y + 30)
                {
                    result = false;
```

```
                }

                else if(_mouseX >= blocPosition.x-20 &&  _mouseX <= blocPosition
.x+_musicButton->get_w()-40
                        && _mouseY >= blocPosition.y && _mouseY <= blocPosition.
y+_musicButton->get_h()-20)
                {
                    if(_music.getStatus() == _music.Playing)
                    {
                        _music.pause();
                    }

                    else
                    {
                        _music.play();
                    }
                }
                else if(_mouseX >= blocSoundPosition.x-20 &&  _mouseX <= blocSou
ndPosition.x+_soundButton->get_w()-40
                        && _mouseY >= blocSoundPosition.y && _mouseY <= blocSoun
dPosition.y+_soundButton->get_h()-20)
                {
                    if(_sounds)
                    {
                        _sounds = false;
                    }

                    else
                    {
                        _sounds = true;
                    }
                }

                _cursor->setTexture(_TextureCursorClicPressed);

            }
            else if(event.type == sf::Event::MouseButtonReleased && event.mouseB
utton.button == sf::Mouse::Left)
            {
                _cursor->setTexture(_TextureCursorClicReleased);
            }

            else if(event.type == sf::Event::KeyPressed && event.key.code == sf:
:Keyboard::Escape)
            {
                result = false;
            }
        }
        if(_mouseX >= -12 && _mouseX <= 10 +_previousPage->get_w()-15
                && _mouseY >= 10 && _mouseY <= 10 + _previousPage->get_h())
        {
            _previousPage->setSelection(true);
            _previousPage->rectColorChange(sf::Color(18,225,225,180));
        }
        else if(_mouseX >= 965 && _mouseX <= 990 +_nextPage->get_w()
                && _mouseY >= 10 && _mouseY <= 10 + _nextPage->get_h())
        {
            _nextPage->setSelection(true);
            _nextPage->rectColorChange(sf::Color(18,225,225,180));
        }
        else if(_mouseX >= _closeInfo.getPosition().x-20 && _mouseX <= _closeInf
o.getPosition().x +400
                && _mouseY >= _closeInfo.getPosition().y -10 && _mouseY <= _clos
eInfo.getPosition().y + 30)
        {
            _closeInfo.setColor(sf::Color(125,130,122,200));
        }
        else
        {
```



```
            _nextPage->setSelection(false);
            _previousPage->setSelection(false);
            _nextPage->rectColorChange(sf::Color(18,140,225,230));
            _previousPage->rectColorChange(sf::Color(18,140,225,230));
            _closeInfo.setColor(sf::Color::Black);
        }
    }
    return result;
}


//=========================================================================
==========================================================================
// Description : Chargement d'images
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de charger les images dans les textures et fait planter le
jeu si une image ne s'est pas chargÃ© (accompagnÃ© d'un message d'erreur)
//=========================================================================
==========================================================================
bool Rules::loadImages()
{
    bool success = true;

    for(int i=0; i<12;i++)
    {
        sf::IntRect r(52*i,0,50,50);
        AnimCoin.push_back(r);
    }
    for(int i=0; i<4;i++)
    {
        sf::IntRect r9(80*i,0,78,50);
        AnimBird.push_back(r9);
    }

    for(int i=0; i<7;i++)
    {
        sf::IntRect r10(150*i,0,148,75);
        AnimWolf.push_back(r10);
    }

    for(int i=0; i<4;i++)
    {
        sf::IntRect r11(100*i,0,99,50);
        AnimEagle.push_back(r11);
    }

    for(int i=0; i<6; i++)
    {
        sf::IntRect r16(110*i,0,110,100);
        AnimCopter.push_back(r16);
    }

    for(int i=0; i<4; i++)
    {
        sf::IntRect r1(75*i,0,73,75);
        AnimYeti.push_back(r1);
    }
    if (!_TextureYeti.loadFromFile(YETI_WALK))
    {
        std::cerr << "ERROR when loading image file: " << YETI_WALK << std::endl
;
        success = false;
    }
    if (!_TextureBird.loadFromFile(BIRD))
    {
        std::cerr << "ERROR when loading image file: " << BIRD << std::endl;
        success = false;
    }
```

```
    if (!_TextureCopter.loadFromFile(HELICO))
    {
        std::cerr << "ERROR when loading image file: " << HELICO << std::endl;
        success = false;
    }
    if (!_TextureBonusNuke.loadFromFile(BONUS_NUKE))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_NUKE << std::end
l;
        success = false;
    }
    if (!_TextureEagle.loadFromFile(EAGLE))
    {
        std::cerr << "ERROR when loading image file: " << EAGLE << std::endl;
        success = false;
    }
    if (!_TextureWolf.loadFromFile(WOLF))
    {
        std::cerr << "ERROR when loading image file: " << WOLF << std::endl;
        success = false;
    }
    if (!_TextureRock.loadFromFile(ROCK))
    {
        std::cerr << "ERROR when loading image file: " << ROCK << std::endl;
        success = false;
    }
    if (!_TextureTree.loadFromFile(TREE))
    {
        std::cerr << "ERROR when loading image file: " << TREE << std::endl;
        success = false;
    }
    if (!_TextureSilverCoin.loadFromFile(SILVER_COIN))
    {
        std::cerr << "ERROR when loading image file: " << SILVER_COIN << std::en
dl;
        success = false;
    }
    if (!_TextureGoldCoin.loadFromFile(GOLD_COIN))
    {
        std::cerr << "ERROR when loading image file: " << GOLD_COIN << std::endl
;
        success = false;
    }
    if (!_TextureBonusLife.loadFromFile(BONUS_LIFE))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_LIFE << std::end
l;
        success = false;
    }
    if (!_TextureBonusInvincible.loadFromFile(BONUS_INVINCIBLE))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_INVINCIBLE << st
d::endl;
        success = false;
    }
    if (!_TextureBonusPoint.loadFromFile(BONUS_POINT))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_POINT << std::en
dl;
        success = false;
    }
    if (!_TextureBonusJump.loadFromFile(BONUS_JUMP))
    {
        std::cerr << "ERROR when loading image file: " << BONUS_JUMP << std::end
l;
        success = false;
    }
    if (!_TextureArrow.loadFromFile(ARROW))
    {
```



```
        std::cerr << "ERROR when loading image file: " << ARROW << std::endl;
        success = false;
    }
    return success;
}
```

```
#ifndef RULES_H
#define RULES_H

#include "View.h"
#include "Button.h"

class Rules : public View
{
private:
    const std::string ARROW = "Images/Boutons/arrow.png";

    sf::RectangleShape _header;
    sf::Text _title;
    std::string _stringTitle;
    sf::RectangleShape _struct;

    //Elements textes utilisÃ©s pour les rÃ¨gles
    sf::Text _subTitle;
    std::string _stringSubTitle;

    sf::Text _subSubTitle;
    std::string _stringSubSubTitle;

    sf::Text _subSubTitle2;
    std::string _stringSubSubTitle2;

    sf::Text _desc1;
    std::string _stringDesc1;

    sf::Text _desc2;
    std::string _stringDesc2;

    sf::Text _desc3;
    std::string _stringDesc3;

    sf::Text _desc4;
    std::string _stringDesc4;

    sf::Text _desc5;
    std::string _stringDesc5;

    sf::Text _desc6;
    std::string _stringDesc6;

    sf::Text _closeInfo;
    std::string _stringCloseInfo;

    //Indique sur quelle page se situe l'utilisateur
    bool _page1 = true;
    bool _page2 = false;
    bool _page3 = false;

    //Boutons pour changer de pages
    Button* _nextPage;
    Button* _previousPage;
    std::string _stringNextButton;
    std::string _stringPreviousButton;

   /// ELEMENTS DES DIFFERENTES PAGES

    // PAGE 1 : JOUEUR ET OBJECTIFS
    AnimatedGraphicElement* _yeti;
    sf::Texture _TextureYeti;
    GraphicElement* _arrow;
    sf::Texture _TextureArrow;

    // PAGE 2 : LES OBSTACLES
    GraphicElement* _tree;
    GraphicElement* _rock;
```



```
    AnimatedGraphicElement* _wolf;
    AnimatedGraphicElement* _bird;
    AnimatedGraphicElement* _eagle;
    AnimatedGraphicElement* _copter;

    std::vector<sf::IntRect> AnimYeti;
    std::vector<sf::IntRect> AnimBird;
    std::vector<sf::IntRect> AnimWolf;
    std::vector<sf::IntRect> AnimEagle;
    std::vector<sf::IntRect> AnimCopter;

    sf::Texture _TextureTree;
    sf::Texture _TextureRock;
    sf::Texture _TextureWolf;
    sf::Texture _TextureEagle;
    sf::Texture _TextureBird;
    sf::Texture _TextureCopter;


    // PAGE 3 : LES BONUS
    GraphicElement* _bonusHealth;
    GraphicElement* _bonusJump;
    GraphicElement* _bonusShield;
    GraphicElement* _bonusNuke;
    GraphicElement* _bonusPoint;
    AnimatedGraphicElement *_silverCoin;
    AnimatedGraphicElement *_goldCoin;

    std::vector<sf::IntRect> AnimCoin;

    sf::Texture _TextureBonusNuke;
    sf::Texture _TextureBonusInvincible;
    sf::Texture _TextureBonusLife;
    sf::Texture _TextureBonusPoint;
    sf::Texture _TextureBonusJump;
    sf::Texture _TextureSilverCoin;
    sf::Texture _TextureGoldCoin;



public:
    Rules(int w, int h, sf::RenderWindow* window);
    ~Rules();
    void draw();
    bool treatEvents();
    void synchronize();
    bool loadImages();
};

#endif // RULES_H
```

```
#include "Settings.h"


//=============================================================================
=======
// Description : Constructeur de Settings
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de construire un ecran d'options
//=============================================================================
=======
Settings::Settings(int w, int h, sf::RenderWindow *window) :
    View{w,h,window}
{
    _statut = SETTINGS;

    _backToMenu = new Button(200,75,sf::Color(18,140,225,230), _stringBackMenu);
    _backToMenu->setPosition(10,10);
    _backToMenu->setTextPosition(32,30);
    _backToMenu->textSize(22);

    _titleBorder = sf::RectangleShape{sf::Vector2f{600,60}};
    _titleBorder.setOutlineColor(sf::Color::Black);
    _titleBorder.setFillColor(sf::Color(18,140,225,150));
    _titleBorder.setOutlineThickness(7);
    _titleBorder.setPosition(300, 30);

    _struct = sf::RectangleShape{sf::Vector2f{800,450}};
    _struct.setOutlineColor(sf::Color::Black);
    _struct.setFillColor(sf::Color(18,140,225,150));
    _struct.setOutlineThickness(2);
    _struct.setPosition(200, 130);

    _title.setFont(_fontTitle);
    _title.setColor(sf::Color::Black);
    _title.setPosition(500,28);
    _title.setCharacterSize(48);

    _difficultyTitle.setFont(_font);
    _difficultyTitle.setColor(sf::Color::White);
    _difficultyTitle.setPosition(300,180);
    _difficultyTitle.setCharacterSize(36);

    _difficultyParam.setFont(_font);
    _difficultyParam.setColor(sf::Color::White);
    _difficultyParam.setPosition(715,180);
    _difficultyParam.setCharacterSize(36);

    _decreasementDifficulty= sf::CircleShape(15, 3);
    _addingDifficulty= sf::CircleShape(15, 3);
    _decreasementDifficulty.setRotation(270);
    _addingDifficulty.setRotation(90);
    _decreasementDifficulty.setPosition(650,220);
    _addingDifficulty.setPosition(910,190);

    _musicInfo.setFont(_font);
    _musicInfo.setColor(sf::Color::White);
    _musicInfo.setPosition(300,280);
    _musicInfo.setCharacterSize(36);

    _musicParam.setFont(_font);
    _musicParam.setColor(sf::Color::White);
    _musicParam.setPosition(715,280);
    _musicParam.setCharacterSize(36);

    _SoundEffectInfo.setFont(_font);
    _SoundEffectInfo.setColor(sf::Color::White);
    _SoundEffectInfo.setPosition(300,380);
    _SoundEffectInfo.setCharacterSize(36);
```



```
    _SoundEffectParam.setFont(_font);
    _SoundEffectParam.setColor(sf::Color::White);
    _SoundEffectParam.setPosition(715,380);
    _SoundEffectParam.setCharacterSize(36);

    _languageInfo.setFont(_font);
    _languageInfo.setColor(sf::Color::White);
    _languageInfo.setPosition(300,480);
    _languageInfo.setCharacterSize(36);

    _languageParam.setFont(_font);
    _languageParam.setColor(sf::Color::White);
    _languageParam.setPosition(715,480);
    _languageParam.setCharacterSize(36);

    _changeDownLanguage= sf::CircleShape(15, 3);
    _changeUpLanguage= sf::CircleShape(15, 3);
    _changeDownLanguage.setRotation(270);
    _changeUpLanguage.setRotation(90);
    _changeDownLanguage.setPosition(650,520);
    _changeUpLanguage.setPosition(910,490);
}

//=============================================================================
==========================================
// Description : Destructeur de Settings
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de dÃ©sallouer toutes les cases mÃ©moires allouÃ©es dynamiq
uement dans le GameView et de supprimer ce dernier
//=============================================================================
==========================================
Settings::~Settings()
{
    delete _backToMenu;
}

//=============================================================================
// Description : Action de Dessin
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : Permet de dessiner tous les elements graphiques sur la fenÃªtre
//=============================================================================
void Settings::draw()
{
    View::draw();
    _backToMenu->draw(_window);
    _window->draw(_titleBorder);
    _window->draw(_title);
    _window->draw(_struct);
    _window->draw(_difficultyTitle);
    _window->draw(_difficultyParam);

    if(_model->getDifficulty() != Easy)
        _window->draw(_decreasementDifficulty);
    if(_model->getDifficulty() != Hard)
        _window->draw(_addingDifficulty);

    _window->draw(_musicInfo);
    _window->draw(_musicParam);
    _window->draw(_SoundEffectInfo);
    _window->draw(_SoundEffectParam);
    _window->draw(_languageInfo);
    _window->draw(_languageParam);

    if(_lang != English)
        _window->draw(_changeDownLanguage);
    if(_lang != German)
```

```
        _window->draw(_changeUpLanguage);

    _cursor->draw(_window);
    _window->display();
}



//============================================================================
========================
// Description : Detecteur et traitement d'Ã©venements
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : permet de rÃ©cupÃ©rer les Ã©venements de l'utilisateur et de les t
raiter en fonction de ceux-ci
//============================================================================
========================
bool Settings::treatEvents()
{
    bool result = false;
    if(_window->isOpen())
    {
        result = true;
        sf::Event event;

        _window_position = (_window->getPosition());
        _mouse_position = (sf::Mouse::getPosition());
        _mouseX = ((_mouse_position.x) - (_window_position.x)) -20;
        _mouseY = ((_mouse_position.y) - (_window_position.y)) +35;

        while (_window->pollEvent(event))
        {
            if(event.type == sf::Event::Closed)
            {
                _window->close();
                result = false;
            }

            if(event.type == sf::Event::MouseButtonPressed && event.mouseButton.
button == sf::Mouse::Left)
            {
                if(_backToMenu->getSelected())
                {
                    result = false;
                    _statut = MENU;
                }

                else if(_mouseX >=_decreasementDifficulty.getPosition().x-25 &&
_mouseX <= _decreasementDifficulty.getPosition().x+25
                        && _mouseY >=_decreasementDifficulty.getPosition().y-25
&& _mouseY <= _decreasementDifficulty.getPosition().y)
                {

                    _model->decreaseDifficulty();
                }

                else if(_mouseX >=_addingDifficulty.getPosition().x-75 && _mouse
X <= _addingDifficulty.getPosition().x
                        && _mouseY >=_addingDifficulty.getPosition().y && _mouse
Y <= _addingDifficulty.getPosition().y+25)
                    _model->addDifficulty();


                else if(_mouseX >= 690 &&  _mouseX <= 760 && _mouseY >= 275 && _
mouseY <= 320)
                {
                    if(_music.getStatus() == _music.Playing)
                        _music.pause();
                    else
```



```
                        _music.play();
                }
                else if(_mouseX >= 690 &&  _mouseX <= 760 && _mouseY >= 375 && _
mouseY <= 420)
                {
                    if(_sounds)
                        _sounds = false;
                    else
                        _sounds = true;
                }
                else if(_mouseX >=_changeUpLanguage.getPosition().x-75 && _mouse
X <= _changeUpLanguage.getPosition().x
                        && _mouseY >=_changeUpLanguage.getPosition().y && _mouse
Y <= _changeUpLanguage.getPosition().y+25)
                {
                    if(_lang == English)
                        _lang = French;
                    else if(_lang == French)
                        _lang = Spanish;
                    else if(_lang == Spanish)
                        _lang = German;
                }

                else if(_mouseX >=_changeDownLanguage.getPosition().x-25 && _mou
seX <= _changeDownLanguage.getPosition().x+25
                        && _mouseY >=_changeDownLanguage.getPosition().y-25 && _
mouseY <= _changeDownLanguage.getPosition().y)
                {
                    if(_lang == German)
                        _lang = Spanish;
                    else if(_lang == French)
                        _lang = English;
                    else if(_lang == Spanish)
                        _lang = French;
                }
                _cursor->setTexture(_TextureCursorClicPressed);
            }
            else if(event.type == sf::Event::MouseButtonReleased && event.mouseB
utton.button == sf::Mouse::Left)
            {
                _cursor->setTexture(_TextureCursorClicReleased);
            }

        }

        if(_mouseX >= -12 && _mouseX <= 10 +_backToMenu->get_w()-15
                && _mouseY >= 10 && _mouseY <= 10 + _backToMenu->get_h())
            _backToMenu->setSelection(true);

        else if(_mouseX >=_decreasementDifficulty.getPosition().x-25 && _mouseX
<= _decreasementDifficulty.getPosition().x+25
                && _mouseY >=_decreasementDifficulty.getPosition().y-25 && _mous
eY <= _decreasementDifficulty.getPosition().y)
            _decreasementDifficulty.setFillColor(sf::Color(200,190,210,180));

        else if(_mouseX >=_addingDifficulty.getPosition().x-75 && _mouseX <= _ad
dingDifficulty.getPosition().x
                && _mouseY >=_addingDifficulty.getPosition().y && _mouseY <= _ad
dingDifficulty.getPosition().y+25)
            _addingDifficulty.setFillColor(sf::Color(200,190,210,180));

        else if(_mouseX >=_changeUpLanguage.getPosition().x-75 && _mouseX <= _ch
angeUpLanguage.getPosition().x
                && _mouseY >=_changeUpLanguage.getPosition().y && _mouseY <= _ch
angeUpLanguage.getPosition().y+25)
            _changeUpLanguage.setFillColor(sf::Color(200,190,210,180));

        else if(_mouseX >=_changeDownLanguage.getPosition().x-25 && _mouseX <= _
changeDownLanguage.getPosition().x+25
```

```
                && _mouseY >=_changeDownLanguage.getPosition().y-25 && _mouseY <
= _changeDownLanguage.getPosition().y)
            _changeDownLanguage.setFillColor(sf::Color(200,190,210,180));


        else if(_mouseX >= 690 &&  _mouseX <= 760 && _mouseY >= 275 && _mouseY <
= 320)
        {
            _musicParam.setColor(sf::Color(200,190,210,180));
        }
        else if(_mouseX >= 690 &&  _mouseX <= 760 && _mouseY >= 375 && _mouseY <
= 420)
        {
            _SoundEffectParam.setColor(sf::Color(200,190,210,180));
        }
        else
        {
            _decreasementDifficulty.setFillColor(sf::Color::White);
            _addingDifficulty.setFillColor(sf::Color::White);
            _changeDownLanguage.setFillColor(sf::Color::White);
            _changeUpLanguage.setFillColor(sf::Color::White);
            _musicParam.setColor(sf::Color::White);
            _SoundEffectParam.setColor(sf::Color::White);
            _backToMenu->setSelection(false);
        }
    }
    return result;
}


//================================================================================
================================================================================
// Description : Synchronisation avec le modÃ¨le et le traitement d'Ã©venements
// Auteur : Guillaume Nedelec, Nicolas Marcilloux
// Date : 20/05/16
// InterÃªt : Permet de mettre Ã  jour l'affiche Ã  l'ecran
//================================================================================
================================================================================
void Settings::synchronize()
{
    View::synchronize();

    if(_backToMenu->getSelected())
        _backToMenu->rectColorChange(sf::Color(18,225,225,180));
    else
        _backToMenu->rectColorChange(sf::Color(18,140,225,230));
    _cursor->setPosition(_mouseX, _mouseY);

    if(_lang == English)
    {
        if(_sounds)
            _stringSoundsEffectsParam = "Yes";
        else
            _stringSoundsEffectsParam = "No";

        if(_music.getStatus() == _music.Playing)
            _stringMusicParam = "Yes";
        else
            _stringMusicParam = "No";

        if(_model->getDifficulty()== Easy)
            _stringDifficultyParam = "Easy";

        else if(_model->getDifficulty()== Medium)
            _stringDifficultyParam = "Medium";

        else if(_model->getDifficulty()== Hard)
            _stringDifficultyParam = "Hard";
```



```
        _stringTitle = "Settings";
        _stringDifficultyTitle = "Difficulty";
        _stringMusic = "Music";
        _stringSoundsEffects = "Sounds Effects";
        _stringLanguage = "Language";
        _stringLanguageParam = "English";
        _stringBackMenu = "Back to menu";
        _backToMenu->setTextPosition(40 ,30);
    }

    else if(_lang == French)
    {
        if(_sounds)
            _stringSoundsEffectsParam = "Oui";
        else
            _stringSoundsEffectsParam = "Non";

        if(_music.getStatus() == _music.Playing)
            _stringMusicParam = "Oui";
        else
            _stringMusicParam = "Non";

        if(_model->getDifficulty()== Easy)
            _stringDifficultyParam = "Facile";

        else if(_model->getDifficulty()== Medium)
            _stringDifficultyParam = "Normal";

        else if(_model->getDifficulty()== Hard)
            _stringDifficultyParam = "Difficile";

        _stringTitle = "Options";
        _stringDifficultyTitle = "Difficulte";
        _stringMusic = "Musique";
        _stringSoundsEffects = "Effets Sonores";
        _stringLanguage = "Langage";
        _stringLanguageParam = "Francais";
        _stringBackMenu = "Retour Menu";
        _backToMenu->setTextPosition(45 ,30);
    }

    else if(_lang == Spanish)
    {
        if(_sounds)
            _stringSoundsEffectsParam = "Si";
        else
            _stringSoundsEffectsParam = "No";

        if(_music.getStatus() == _music.Playing)
            _stringMusicParam = "Si";
        else
            _stringMusicParam = "No";

        if(_model->getDifficulty()== Easy)
            _stringDifficultyParam = "Facil";

        else if(_model->getDifficulty()== Medium)
            _stringDifficultyParam = "Normal";

        else if(_model->getDifficulty()== Hard)
            _stringDifficultyParam = "Dificil";

        _stringTitle = "Opciones";
        _stringDifficultyTitle = "Dificultad";
        _stringMusic = "Musica";
        _stringSoundsEffects = "Efectos Sonoros";
        _stringLanguage = "Idioma";
        _stringLanguageParam = "Espanol";
        _stringBackMenu = "Volver al Menu";
```

```
        _backToMenu->setTextPosition(35 ,30);
    }

    else if(_lang == German)
    {
        if(_sounds)
            _stringSoundsEffectsParam = "Ja";
        else
            _stringSoundsEffectsParam = "Nein";

        if(_music.getStatus() == _music.Playing)
            _stringMusicParam = "Ja";
        else
            _stringMusicParam = "Nein";

        if(_model->getDifficulty()== Easy)
            _stringDifficultyParam = "Leicht";

        else if(_model->getDifficulty()== Medium)
            _stringDifficultyParam = "Mittel";

        else if(_model->getDifficulty()== Hard)
            _stringDifficultyParam = "Schwer";

        _stringTitle = "Optionen";
        _stringDifficultyTitle = "Schwierigkeit";
        _stringMusic = "Musik";
        _stringSoundsEffects = "Soundeffekte";
        _stringLanguage = "Sprache";
        _stringLanguageParam = "Deutsch";
        _stringBackMenu = "Zuruck zum Menu";
        _backToMenu->setTextPosition(20 ,30);
    }

    _title.setString(_stringTitle);
    _difficultyTitle.setString(_stringDifficultyTitle);
    _difficultyParam.setString(_stringDifficultyParam);
    _musicInfo.setString(_stringMusic);
    _musicParam.setString(_stringMusicParam);
    _SoundEffectInfo.setString(_stringSoundsEffects);
    _SoundEffectParam.setString(_stringSoundsEffectsParam);
    _languageInfo.setString(_stringLanguage);
    _languageParam.setString(_stringLanguageParam);

    _backToMenu->setText(_stringBackMenu);
}


//=============================================================================
==
// Description : Mutateur de modÃ¨le
// Auteur : Guillaume Nedelec
// Date : 20/05/16
// InterÃªt : permet de charger de le modÃ¨le courant (celui construit dans le m
ain)
//=============================================================================
==
void Settings::setModel(Model *model)
{
    _model = model;
}
```