```
#include "View.h"
#include "Model.h"
#include "Ninja.h"
#include "GraphicElement.h"
#include "Animatedgraphicelement.h"

#include <sstream>
#include <iostream>
#include <algorithm>
#include <fstream>
using namespace std;

/*
 * ----------------------------------------------------------------------------
 * "THE BEER-WARE LICENSE" (Revision 42):
 * Antoine BOULANGÃM-^I & Pierre_Elliot CABRERA wrote this file. As long as you
retain this notice you
 * can do whatever you want with this stuff. If we meet some day, and you think
 * this stuff is worth it, you can buy me a beer in return
 * ----------------------------------------------------------------------------
 */

//=======================================
// Constructeur
//=======================================
View::View(int w, int h)
    : _w(w), _h(h)
    , _positionScore{0}
    , _language{ENGLISH}
    , _first{'A'}, _second{'A'}, _third{'A'}
{
    // dÃ©finit une fenÃªtre en spÃ©cifiant sa taille, son titre, et le style du
 bouton pour la fermer
    _window = new sf::RenderWindow(sf::VideoMode(w, h, 32), WINDOW_TITLE, sf::St
yle::Close);

    if (!_backgroundArriere.loadFromFile(BACKGROUND_ARRIERE_IMAGE))
        std::cerr << "ERROR when loading image file: " << BACKGROUND_ARRIERE_IMA
GE << std::endl;
    else {
        _slidingBackgroundArriere = new SlidingBackground(_backgroundArriere, w,
 h, 1);
        _slidingBackgroundArriereMenu = new SlidingBackground(_backgroundArriere
, w, h, 1, true);
    }
    if (!_backgroundAvant.loadFromFile(BACKGROUND_AVANT_IMAGE))
        std::cerr << "ERROR when loading image file: " << BACKGROUND_AVANT_IMAGE
 << std::endl;
    else {
        _slidingBackgroundAvant = new SlidingBackground(_backgroundAvant, w, h,
2);
        _slidingBackgroundAvantMenu = new SlidingBackground(_backgroundAvant, w,
 h, 2, true);
    }
    if (!_hpBarTexture.loadFromFile(HP_BAR_IMAGE))
        std::cerr << "ERROR when loading image file: " << HP_BAR_IMAGE << std::e
ndl;
    else
        _hpBar = new GraphicElement(_hpBarTexture, 20, 550, 160, 40);

    if (!_containerTexture.loadFromFile(HP_CONTAINER_IMAGE))
        std::cerr << "ERROR when loading image file: " << HP_CONTAINER_IMAGE <<
std::endl;
    else{
        _container = new GraphicElement(_containerTexture, 168, 48);
    }
    if (!_textureIntroduction.loadFromFile(INTRODUCTION_IMAGE))
        std::cerr << "ERROR when loading image file: " << INTRODUCTION_IMAGE <<
std::endl;
```



```
    else
        _introduction = new GraphicElement(_textureIntroduction, 0, 0, w, h, 1,
1);

    _fontScore = new sf::Font();
    _fontScore->loadFromFile(FONT_SCORE);
    _textScore.setFont(*_fontScore);
    _textScore.setPosition(1000, 550);
    _textScore.setColor(sf::Color::White);

    _fontClassic = new sf::Font();
    _fontClassic->loadFromFile(FONT_CLASSIC);
    _textClassic.setFont(*_fontClassic);
    _textClassic.setColor(sf::Color::White);

    if (!_textureButtonMenu.loadFromFile(BOUTON_MENU_IMAGE))
        std::cerr << "ERROR when loading image file: " << BOUTON_MENU_IMAGE << s
td::endl;
    else{
        std::string namePlay = "Play";
        std::string nameQuit = "Quit";
        std::string nameHighscores = "Highscores";
        std::string nameBack = "Back";
        switch(_language){
        case FRENCH:
            namePlay = "Jouer";
            nameQuit = "Quitter";
            nameHighscores = "Meilleurs scores";
            nameBack = "Retour";
            break;
        default:
            break;
        }
        std::vector <sf::IntRect> clipRects;
        sf::IntRect rect;
        rect.height = 63;
        rect.width = 134;
        rect.top = 0;
        rect.left = 0;
        for (int i = 0; i<2; i++)
        {
            clipRects.push_back(rect);
            rect.left += rect.width;
        }
        _play = new ClickableElement(clipRects, _textureButtonMenu, (w/2) - (BUT
TON_WIDTH/2), (h/2) - BUTTON_HEIGHT, BUTTON_WIDTH, BUTTON_HEIGHT, 2, 1, PLAY, na
mePlay, _fontClassic, BUTTON_HEIGHT/2, BUTTON_HEIGHT/6);
        _quit = new ClickableElement(clipRects, _textureButtonMenu, (w/2) - (BUT
TON_WIDTH/2), (h/2) + BUTTON_HEIGHT/2, BUTTON_WIDTH, BUTTON_HEIGHT, 2, 1, QUIT,
nameQuit, _fontClassic, BUTTON_HEIGHT/2, BUTTON_HEIGHT/6);
        _highscore = new ClickableElement(clipRects, _textureButtonMenu, 200, 20
0, BUTTON_WIDTH, BUTTON_HEIGHT, 2, 1, HIGHSCORES, nameHighscores, _fontClassic,
BUTTON_HEIGHT/4, BUTTON_HEIGHT/3);
        _back = new ClickableElement(clipRects, _textureButtonMenu, 500, 450, BU
TTON_WIDTH, BUTTON_HEIGHT, 2, 1, BACK, nameBack, _fontClassic, BUTTON_HEIGHT/4,
BUTTON_HEIGHT/3);
    }
    if (!_textureSound.loadFromFile(BOUTON_MENU_SOUND))
        std::cerr << "ERROR when loading image file: " << BOUTON_MENU_SOUND << s
td::endl;
    else{
        std::vector <sf::IntRect> clipRects;
        sf::IntRect rect;
        rect.height = 37;
        rect.width = 39;
        rect.top = 0;
        rect.left = 0;
        for (int i = 0; i<2; i++)
        {
```

```
            clipRects.push_back(rect);
            rect.left += rect.width;
        }
        _sound = new ClickableElement(clipRects, _textureSound, 10, 10, 50, 50,
2, 1, SOUND, "", _fontClassic,  0, 0);
    }

    if (!_textureNinja.loadFromFile(NINJA_IMAGE))
        std::cerr << "ERROR when loading image file: " << NINJA_IMAGE << std::en
dl;
    else {
        std::vector <sf::IntRect> clipRects;
        sf::IntRect rect;
        rect.height = 200;
        rect.width = 145;
        rect.top = 0;
        rect.left = 0;
        for (int i = 0; i<10; i++)
        {
            clipRects.push_back(rect);
            rect.left += rect.width;
        }
        _ninjaSprite = new AnimatedGraphicElement(clipRects, _textureNinja, 0, 0
, 54, 75, 10, 8, 5);
    }
    if(!_textureExplosion.loadFromFile(EXPLOSION_IMAGE))
        std::cerr << "ERROR when loading image file: " << EXPLOSION_IMAGE << std
::endl;
    else {
        std::vector <sf::IntRect> clipRectstmp;
        sf::IntRect recttmp;
        recttmp.height = 139;
        recttmp.width = 179;
        recttmp.top = 0;
        recttmp.left = 100;
        for (int i = 0; i<6; i++)
        {
            clipRectstmp.push_back(recttmp);
            recttmp.left += recttmp.width;
        }
        _explosionSpriteTall = new AnimatedGraphicElement(clipRectstmp, _texture
Explosion, 0, 0, 100, 100, 6, 1, 5);
        _explosionSpriteShort = new AnimatedGraphicElement(clipRectstmp, _textur
eExplosion, 0, 0, 75, 75, 6, 1, 5);
        _explosionSpriteFlying = new AnimatedGraphicElement(clipRectstmp, _textu
reExplosion, 0, 0, 50, 50, 6, 1, 5);
    }
    if (!_textureObstacleTall.loadFromFile(OBSTACLE_TALL_IMAGE))
        std::cerr << "ERROR when loading image file: " << OBSTACLE_TALL_IMAGE<<
std::endl;
    if (!_textureObstacleShort.loadFromFile(OBSTACLE_SHORT_IMAGE))
        std::cerr << "ERROR when loading image file: " << OBSTACLE_SHORT_IMAGE <
< std::endl;
    if (!_textureObstacleFlying.loadFromFile(OBSTACLE_FLYING_IMAGE))
        std::cerr << "ERROR when loading image file: " << OBSTACLE_FLYING_IMAGE
<< std::endl;
    if (!_textureBonus.loadFromFile(BONUS_IMAGE))
        std::cerr << "ERROR when loading image file: " << BONUS_IMAGE << std::en
dl;

    if(!_music.openFromFile(BACKGROUND_MUSIC))
        std::cerr << "ERROR when loading image file: " << BACKGROUND_MUSIC << st
d::endl;

    _music.setLoop(true);
    _music.play();

    _clockGeneral.restart();
    _elapsedEvolution = sf::seconds(ELAPSED_TIME_GAME_EVOLUTION);
```



```
    std::vector<ClickableElement *> clickableElements;
    clickableElements.push_back(_play);
    clickableElements.push_back(_quit);
    clickableElements.push_back(_sound);
    clickableElements.push_back(_highscore);
    _menuSlidingPrincipal = new MenuSliding(_slidingBackgroundAvantMenu, _slidin
gBackgroundArriereMenu, 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, clickableElements);

    clickableElements.clear();
    clickableElements.push_back(_back);
    _menuSlidingHighscore = new MenuSliding(_slidingBackgroundAvantMenu, _slidin
gBackgroundArriereMenu, 0, 0, SCREEN_WIDTH, SCREEN_HEIGHT, clickableElements);

    srand(time(NULL));
}

//=======================================
// Destructeur
//=======================================
View::~View() {
    delete _window;
    delete _ninjaSprite;
    delete _play;
    delete _quit;
    delete _highscore;
    delete _back;
    delete _sound;
    delete _introduction;
    delete _explosionSpriteTall;
    delete _explosionSpriteShort;
    delete _explosionSpriteFlying;
    delete _slidingBackgroundArriere;
    delete _slidingBackgroundAvant;
    delete _fontScore;
    delete _fontClassic;
    delete _hpBar;
    delete _container;
    delete _menuSlidingPrincipal;
    delete _slidingBackgroundArriereMenu;
    delete _slidingBackgroundAvantMenu;
    delete _menuSlidingHighscore;

    for(auto it = _elementToGraphicElement.begin(); it != _elementToGraphicEleme
nt.end(); ++it){
        delete (*it).second;
    }

    for(auto it = _elementToAnimatedGraphicElement.begin(); it != _elementToAnim
atedGraphicElement.end(); ++it){
        delete (*it).second;
    }
}

//=======================================
// Fonction de MAJ
//=======================================
void View::synchronize(){
    // Synchronise la position du ninja et les sprites associÃ©s
    int x, y, mvtNinjaX;
    string s;
    _model->getNinjaPosition(x, y);
    _model->getNinjaSpeedX(mvtNinjaX);
    _ninjaSprite->synchronize(x, y);

    // Synchronise le score et son affichage
    s = std::to_string(_model->getScore());
    _textScore.setString(s);
    std::vector <sf::IntRect> clipRects;
```

```
    sf::IntRect rect;
    rect.height = 50;
    rect.width = 50;
    rect.left = 0;

    // CrÃ©e une transposition graphique pour chaque nouvel Ã©lÃ¨ment
    std::vector<const AutonomousElement *> newElements = _model->getNewAutonomou
sElements();
    for(auto itNewElement : newElements)
    {
        GraphicElement * newGraphicElement;
        switch(itNewElement->getType()){
        case BIG:
            newGraphicElement = new GraphicElement(_textureObstacleTall,
                                                   (*itNewElement).getX(),
                                                   (*itNewElement).getY(),
                                                   (*itNewElement).getW(),
                                                   (*itNewElement).getH());
            break;
        case SMALL:
            newGraphicElement = new GraphicElement(_textureObstacleShort,
                                                   (*itNewElement).getX(),
                                                   (*itNewElement).getY(),
                                                   (*itNewElement).getW(),
                                                   (*itNewElement).getH());
            break;
        case FLYING:
            newGraphicElement = new GraphicElement(_textureObstacleFlying,
                                                   (*itNewElement).getX(),
                                                   (*itNewElement).getY(),
                                                   (*itNewElement).getW(),
                                                   (*itNewElement).getH());
            break;
        case INVINCIBILITY:
            rect.top = 100;
            for (int i = 0; i<5; i++)
            {
                clipRects.push_back(rect);
                rect.left += rect.width;
            }
            newGraphicElement = new AnimatedGraphicElement(clipRects,
                                                           _textureBonus,
                                                           (*itNewElement).getX(
),
                                                           (*itNewElement).getY(
),
                                                           (*itNewElement).getW(
),
                                                           (*itNewElement).getH(
),
                                                           5, 4, 5);
            break;
        case FLY:
            rect.top = 150;
            for (int i = 0; i<5; i++)
            {
                clipRects.push_back(rect);
                rect.left += rect.width;
            }
            newGraphicElement = new AnimatedGraphicElement(clipRects,
                                                           _textureBonus,
                                                           (*itNewElement).getX(
),
                                                           (*itNewElement).getY(
),
                                                           (*itNewElement).getW(
),
                                                           (*itNewElement).getH(
),
```



```
                                                           5, 4, 5);
            break;
        case SCORE:
            rect.top = 0;
            for (int i = 0; i<5; i++)
            {
                clipRects.push_back(rect);
                rect.left += rect.width;
            }
            newGraphicElement = new AnimatedGraphicElement(clipRects,
                                                           _textureBonus,
                                                           (*itNewElement).getX(
),
                                                           (*itNewElement).getY(
),
                                                           (*itNewElement).getW(
),
                                                           (*itNewElement).getH(
),
                                                           5, 4, 5);
            break;
        default:
        case HEAL:
            rect.top = 50;
            for (int i = 0; i<5; i++)
            {
                clipRects.push_back(rect);
                rect.left += rect.width;
            }
            newGraphicElement = new AnimatedGraphicElement(clipRects,
                                                           _textureBonus,
                                                           (*itNewElement).getX(
),
                                                           (*itNewElement).getY(
),
                                                           (*itNewElement).getW(
),
                                                           (*itNewElement).getH(
),
                                                           5, 4, 5);
            break;
        }
        _elementToGraphicElement[itNewElement] = newGraphicElement;
    }

    // CrÃ©e une transposition graphique animÃ©e pour chaque nouvel Ã©lÃ¨ment dÃ
©truit
    std::vector<const AutonomousElement *> newElementsDeleted = _model->getNewEl
ementsDeleted();
    for(auto itNewElement : newElementsDeleted)
    {
        AnimatedGraphicElement * newAnimatedGraphicElement;
        if(itNewElement->getType() < NUMBER_OBSTACLE){
            switch((*itNewElement).getType()){
            case BIG: // Grand obstacle
                newAnimatedGraphicElement = new AnimatedGraphicElement(_explosio
nSpriteTall, 6, 1);
                break;
            case SMALL: // Petit obstacle
                newAnimatedGraphicElement = new AnimatedGraphicElement(_explosio
nSpriteShort, 6, 1);
                break;
            case FLYING: // Obstacle volant
                newAnimatedGraphicElement = new AnimatedGraphicElement(_explosio
nSpriteFlying, 6, 1);
                break;
            default:
                break;
            }
```

```
            _elementToAnimatedGraphicElement[itNewElement] = newAnimatedGraphicE
lement;
        }
    }

    // Synchronize les positions des Ã©lÃ¨ments toujours existants
    // Si les Ã©lÃ¨ments n'existent plus, ils sont placÃ©s dans un tableau
    // dont tous les objets sont supprimÃ©s Ã  la suite
    std::vector<const AutonomousElement *> elementsToDelete;
    for(auto itElement : _elementToGraphicElement){
        std::vector<AutonomousElement *> elements = _model->getAutonomousElement
s();
        if(find(elements.begin(), elements.end(), itElement.first) != elements.e
nd())
            itElement.second->synchronize(itElement.first->getX(), itElement.fir
st->getY());
        else{
            elementsToDelete.push_back(itElement.first);
        }
    }
    for(auto it : elementsToDelete){
        delete _elementToGraphicElement[it];
        _elementToGraphicElement.erase(it);
    }

    // Synchronize les positions des Ã©lÃ¨ments animÃ©s (representant l'explosio
n) toujours existants
    // Si les Ã©lÃ¨ments n'existent plus, ils sont placÃ©s dans un tableau
    // dont tous les objets sont supprimÃ©s Ã  la suite
    std::vector<const AutonomousElement *> elementsAnimatedToDelete;
    for(auto itElement : _elementToAnimatedGraphicElement){
        std::vector<AutonomousElement *> elementsDeleted = _model->getElementsDe
leted();
        if(find(elementsDeleted.begin(), elementsDeleted.end(), itElement.first)
 != elementsDeleted.end()){
            if (itElement.second->getCounterDestruction() > itElement.second->ge
tLimitDestruction()){
                elementsAnimatedToDelete.push_back(itElement.first);
            }else
                itElement.second->synchronize(itElement.first->getX()-itElement.
first->getW(),
                                              (itElement.first->getY()+itElement
.first->getH())-itElement.second->getH());
        }
        else{
            elementsAnimatedToDelete.push_back(itElement.first);
        }
    }
    for(auto it : elementsAnimatedToDelete){
        delete _elementToAnimatedGraphicElement[it];
        _elementToAnimatedGraphicElement.erase(it);
    }

    // Reset les tableaux de nouveaux Ã©lÃ¨ments
    _model->clearNewElements();
    _model->clearNewElementsDeleted();

    // A chaque certain montant de temps Ã©coulÃ©, augmente la vitesse du jeu
    if (_clockGeneral.getElapsedTime() > _elapsedEvolution)
    {
        _slidingBackgroundArriere->addSpeed(1);
        _slidingBackgroundAvant->addSpeed(1);
        _model->addSpeed();
        _clockGeneral.restart();
    }
}

//=======================================
// Dessine le fond colorÃ©
```



```
//=======================================
void View::drawColoredBackground(){
    // L'utilisation de sf::VertexArray permet au fond d'Ãªtre un dÃ©gradÃ©
    sf::VertexArray coloredBackground(sf::Quads, 4);
    coloredBackground[0].position = sf::Vector2f(0, 0);
    coloredBackground[1].position = sf::Vector2f(SCREEN_WIDTH, 0);
    coloredBackground[2].position = sf::Vector2f(SCREEN_WIDTH, SCREEN_HEIGHT);
    coloredBackground[3].position = sf::Vector2f(0, SCREEN_HEIGHT);

    // Moins le joueur possÃ¨de de point de vie, plus le fond rougit
    coloredBackground[0].color = coloredBackground[1].color = sf::Color(255, 0,
0, 255-((255/8)*_model->getHp()));
    coloredBackground[2].color = coloredBackground[3].color = sf::Color(0, 0, 0,
 128);

    _window->draw(coloredBackground);
}

//=======================================
// Dessine la barre de points de vie en fonction
// des points de vie restant et son 'conteneur'
//=======================================
void View::drawHp(){
    sf::RectangleShape hpBackground;

    hpBackground.setFillColor(sf::Color(28, 30, 39, 255));
    hpBackground.setPosition(sf::Vector2f(_hpBar->getX(), _hpBar->getY()));
    hpBackground.setSize(sf::Vector2f(160, 40));
    _window->draw(hpBackground);

    _hpBar->setSize(160 - (20*(8-_model->getHp())), 40);
    _hpBar->draw(_window);

    _container->setPosition(sf::Vector2f(_hpBar->getX() - 4, _hpBar->getY() - 4)
);
    _container->draw(_window);
}

void View::draw() {
    _window->clear();
    if (_model->getIntroduction()){
        _introduction->draw(_window);
    }
    if (_model->getMenu()){
        // Affiche le menu de dÃ©but
        if (!_model->getMenuHighscore())
            _menuSlidingPrincipal->draw(_window);
        else{
            readFile();
            sf::Text scores;
            scores.setFont(*_fontClassic);
            scores.setPosition(100, 100);
            scores.setColor(sf::Color::White);
            scores.setCharacterSize(100);
            std::string s = "HIGHSCORE";
            scores.setString(s);
            _window->draw(scores);

            scores.setPosition(100, 200);
            s = _highscore1.first + " : " + to_string(_highscore1.second);
            scores.setString(s);
            _window->draw(scores);

            scores.setPosition(100, 300);
            s = _highscore2.first + " : " + to_string(_highscore2.second);;
            scores.setString(s);
            _window->draw(scores);

            s = _highscore3.first + " : " + to_string(_highscore3.second);;
```

```
            scores.setPosition(100, 400);
            scores.setRotation(0);
            scores.setString(s);
            _window->draw(scores);
            _menuSlidingHighscore->draw(_window);
        }
    }
    else if (!_model->getMenu() && !_model->getIntroduction()){
        if (!_model->getDead()){
            drawColoredBackground();
            // Si le joueur subit des dÃ©gats, un rendu visuel est effectuÃ© par
 le passage au blanc du fond
            if(_model->getDamaged()){
                sf::RectangleShape fond;
                fond.setSize(sf::Vector2f(1200, 600));
                fond.setPosition(sf::Vector2f(0, 0));
                fond.setFillColor(sf::Color::White);
                _window->draw(fond);
                _model->changeDamaged();
            }

            // Dessine les fonds animÃ©s
            _slidingBackgroundArriere->draw(_window);
            _slidingBackgroundAvant->draw(_window);

            // Affiche le score
            _window->draw(_textScore);

            // Dessine les Ã©lements de la map
            for(auto it = _elementToGraphicElement.begin(); it != _elementToGrap
hicElement.end(); ++it)
                it->second->draw(_window);

            // Dessine les Ã©lements animÃ©s (explosions)
            for(auto it = _elementToAnimatedGraphicElement.begin(); it != _eleme
ntToAnimatedGraphicElement.end(); ++it){
                it->second->draw(_window);
            }

            // Dessine la ninja
            int mvtNinjaX;
            _model->getNinjaSpeedX(mvtNinjaX);
            _ninjaSprite->drawNinja(_window, mvtNinjaX, _model->getNinjaStatus()
, _model->getInvincibility());

            // Dessine les points de vie
            drawHp();
        }else if (_model->getDead() && !_model->getHighscore()){
            sf::Text textGameOver;
            textGameOver.setFont(*_fontClassic);
            textGameOver.setPosition(300, 100);
            textGameOver.setColor(sf::Color(153, 0, 0));

            std::string s = "Game";
            textGameOver.setString(s);
            textGameOver.setCharacterSize(200);

            _window->draw(textGameOver);

            textGameOver.setPosition(500, 150);
            textGameOver.setColor(sf::Color::White);

            s = "Over";
            textGameOver.setString(s);
            textGameOver.setRotation(15);
            _window->draw(textGameOver);

            s = "<";
            textGameOver.setPosition(380, 210);
```



```
            textGameOver.setRotation(0);
            textGameOver.setString(s);
            _window->draw(textGameOver);
        }
        else if (_model->getDead() && _model->getHighscore()){
            registerHighscore();
        }
    }

    // affiche le tout
    _window->display();
    if (_model->getDead() && !_model->getHighscore() && !_model->getMenu()){
        sf::sleep(sf::seconds(3.f));
        _model->setMenu(false);
        _model->setIntroduction(false);
        _model->setHighscore(true);
    }
    else if (_model->getIntroduction()){
        sf::sleep(sf::seconds(5.f));
        _model->setIntroduction(false);
        _model->setMenu(true);
    }
}

//=====================================
// Enregistre les meilleurs scores grÃ¢ce
// Ã  un identifiant de 3 char
//=====================================
void View::registerHighscore(){
    sf::Text textGameOver;
    textGameOver.setFont(*_fontClassic);
    textGameOver.setCharacterSize(150);
    textGameOver.setStyle(sf::Text::Underlined);

    textGameOver.setPosition(400, 100);
    textGameOver.setString(_first);
    if (_positionScore == 0)
        textGameOver.setColor(sf::Color::Red);
    else
        textGameOver.setColor(sf::Color::White);
    _window->draw(textGameOver);

    textGameOver.setPosition(600, 100);
    textGameOver.setString(_second);
    if (_positionScore == 1)
        textGameOver.setColor(sf::Color::Red);
    else
        textGameOver.setColor(sf::Color::White);
    _window->draw(textGameOver);

    textGameOver.setPosition(800, 100);
    textGameOver.setString(_third);
    if (_positionScore == 2)
        textGameOver.setColor(sf::Color::Red);
    else
        textGameOver.setColor(sf::Color::White);
    _window->draw(textGameOver);
    _window->display();
}

//=====================================
// Lecture dans un fichier
//=====================================
void View::readFile(){
    fstream fichier;
    fichier.open(HIGHSCORE_FILE.c_str(), ios::in );
    if(fichier.fail() )
        cerr << "ouverture en lecture " << HIGHSCORE_FILE << " impossible" << en
dl;
```

```
    else{
        int money;
        std::string name;
        int score;
        fichier >> money;
        _model->setMoney(money);

        fichier >> name;
        fichier >> score;
        _highscore1 = std::make_pair(name, score);

        fichier >> name;
        fichier >> score;
        _highscore2 = std::make_pair(name, score);

        fichier >> name;
        fichier >> score;
        _highscore3 = std::make_pair(name, score);
        fichier.close();
    }
}

//=====================================
// Met Ã  jour les meilleurs scores
//=====================================
void View::synchronizeHighscores(std::string name){
    int score = _model->getScore();
    if(_highscore1.second < score){
        _highscore3 = _highscore2;
        _highscore2 = _highscore1;
        _highscore1 = std::make_pair(name, score);
    }else if(_highscore2.second < score){
        _highscore3 = _highscore2;
        _highscore2 = std::make_pair(name, score);
    }else if(_highscore3.second < score)
        _highscore3 = std::make_pair(name, score);
}

//=====================================
// Ecriture dans un fichier
//=====================================
void View::writeFile(std::string name){
    fstream fichier;
    fichier.open(HIGHSCORE_FILE.c_str(), ios::out);
    if( fichier.fail() )
        cerr << "ouverture en Ã©criture " << HIGHSCORE_FILE << " impossible" <<
endl;
    else{
        synchronizeHighscores(name);
        fichier << _model->getMoney() << endl;
        fichier << _highscore1.first << " " << _highscore1.second << endl;
        fichier << _highscore2.first << " " << _highscore2.second << endl;
        fichier << _highscore3.first << " " << _highscore3.second << endl;
        fichier.close();
    }
}

//=====================================
// Traitement des evenements
//=====================================
bool View::treatEvents() {
    bool result = false;
    if (_window->isOpen()) {
        result = true;
        sf::Event event;
        while (_window->pollEvent(event)) { // parcours les Ã©vÃ©nements de la f
enÃªtre

            if ((event.type == sf::Event::Closed) || // si l'Ã©vÃ©nement ferme l
```



```
a fenÃªtre
                ((event.type == sf::Event::KeyPressed) && (event.key.code ==
 sf::Keyboard::Escape))) {
                _window->close(); // fermeture fenÃªtre
                result = false;
                _model->setMenu(false);
            }
            else if (_model->getMenu() && result && !_model->getDead()){
                if (!_model->getMenuHighscore()){
                    if (event.type == sf::Event::MouseButtonPressed)
                    {
                        if (event.mouseButton.button == sf::Mouse::Left)
                        {
                            int id = NONE;
                            auto clickableElements = _menuSlidingPrincipal->getC
lickableElements();
                            auto it = clickableElements.begin();
                            while (it != clickableElements.end() && id == NONE){
                                id = (*it)->over(event.mouseButton.x, event.mous
eButton.y);
                                ++it;
                            }
                            switch (id) {
                            case PLAY:
                                _model->setMenu(false);
                                break;
                            case QUIT:
                                result = false;
                            case SOUND:
                                if (_sound->getChecked()){
                                    _sound->ChangeSprite(SOUND);
                                    _sound->InverseChecked();
                                    _music.pause();
                                }
                                else{
                                    _sound->ChangeSprite(NONE);
                                    _sound->InverseChecked();
                                    _music.play();
                                }
                                break;
                            case HIGHSCORES:
                                _model->setMenuHighScore(true);
                                break;
                            default:
                                break;
                            }
                        }
                    }

                    auto clickableElements = _menuSlidingPrincipal->getClickable
Elements();
                    auto it = clickableElements.begin();
                    while (it != clickableElements.end()){
                        if ((*it)->getId() != SOUND)
                            (*it)->ChangeSprite((*it)->over(sf::Mouse::getPositi
on(*_window).x,sf::Mouse::getPosition(*_window).y));
                        ++it;
                    }
                }
                else{
                    if (event.type == sf::Event::MouseButtonPressed){
                        if (event.mouseButton.button == sf::Mouse::Left){
                            int id = NONE;
                            auto clickableElements = _menuSlidingHighscore->getC
lickableElements();
                            auto it = clickableElements.begin();
                            while (it != clickableElements.end() && id == NONE){
                                id = (*it)->over(event.mouseButton.x, event.mous
eButton.y);
```

```
                                ++it;
                            }
                            switch (id) {
                            case BACK:
                                _model->setMenuHighScore(false);
                                break;
                            default:
                                break;
                            }
                        }

                    }
                }
                auto clickableElements = _menuSlidingHighscore->getClickableElem
ents();
                auto it = clickableElements.begin();
                while (it != clickableElements.end()){
                    if ((*it)->getId() != SOUND)
                        (*it)->ChangeSprite((*it)->over(sf::Mouse::getPosition(*
_window).x,sf::Mouse::getPosition(*_window).y));
                    ++it;
                }
            }
            else if(!_model->getMenu() && result && !_model->getDead() && !_mode
l->getHighscore()){
                if (event.type == sf::Event::KeyPressed)
                {
                    switch (event.key.code)
                    {
                    case sf::Keyboard::Left:
                        _model->setNinjaToChangeDirection(LEFT);

                        if(_model->getNinjaStatus() != JUMPING){
                            _model->moveNinjaX(false);
                            _model->setNinjaDirection(false);
                        }else
                            _model->setNinjaReduction(true);
                        break;

                    case sf::Keyboard::Right:
                        _model->setNinjaToChangeDirection(RIGHT);

                        if(_model->getNinjaStatus() != JUMPING){
                            _model->moveNinjaX(true);
                            _model->setNinjaDirection(true);
                        }else
                            _model->setNinjaReduction(true);
                        break;

                    case sf::Keyboard::Up:
                    case sf::Keyboard::Space:
                        switch(_model->getNinjaStatus()){
                        case WALKING:
                            _model->moveNinjaY(true);
                            int x, y;
                            _model->getNinjaPosition(x, y);
                            _model->setNinjaXBeginJmp(x);
                            _model->setNinjaYBeginJmp(y);
                            _model->setNinjaStatus(JUMPING);
                            break;
                        case SOARING:
                            _model->moveNinjaY(true);
                            break;
                        default:
                            break;
                        }
                        break;

                    case sf::Keyboard::Down:
```



```
                        if(_model->getNinjaStatus() == SOARING)
                            _model->moveNinjaY(false);
                        break;

                    default:
                        break;
                    }
                }
                else if (event.type == sf::Event::KeyReleased)
                {
                    switch (event.key.code)
                    {
                    case sf::Keyboard::Left:
                    case sf::Keyboard::Right:
                        _model->setNinjaToStopX(true);
                        break;

                    case sf::Keyboard::Up:
                    case sf::Keyboard::Space:
                    case sf::Keyboard::Down:
                        if(_model->getNinjaStatus() != TRANSITIONNING)
                            _model->setNinjaToStopY(true);
                        break;

                    default:
                        break;
                    }
                }
            }
            else if (!_model->getMenu() && result && _model->getDead() && _model
->getHighscore()){
                cout << "coucou" << endl;
                std::string player;
                if (event.type == sf::Event::KeyPressed){
                    switch(event.key.code)
                    {
                    case sf::Keyboard::Up:
                        if (_positionScore == 0){
                            if (_first < 'Z')
                                _first = static_cast<char>(_first+1);
                            else
                                _first = 'A';
                        }
                        if (_positionScore == 1){
                            if (_second < 'Z')
                                _second = static_cast<char>(_second+1);
                            else
                                _second = 'A';
                        }
                        if (_positionScore == 2){
                            if (_third < 'Z')
                                _third = static_cast<char>(_third+1);
                            else
                                _third = 'A';
                        }
                        break;
                    case sf::Keyboard::Down:
                        if (_positionScore == 0){
                            if (_first > 'A')
                                _first = static_cast<char>(_first-1);
                            else
                                _first = 'Z';
                        }
                        if (_positionScore == 1){
                            if (_second > 'A')
                                _second = static_cast<char>(_second-1);
                            else
                                _second = 'Z';
                        }
```

```
                        if (_positionScore == 2){
                            if (_third > 'A')
                                _third = static_cast<char>(_third-1);
                            else
                                _third = 'Z';
                        }
                        break;
                    case sf::Keyboard::Right:
                        _positionScore++;
                        if (_positionScore > 2)
                            _positionScore = 0;
                        break;
                    case sf::Keyboard::Left:
                        _positionScore--;
                        if (_positionScore < 0)
                            _positionScore = 2;
                        break;
                    case sf::Keyboard::Return:
                        readFile();
                        player = _first;
                        player += _second;
                        player += _third;
                        writeFile(player);
                        result = false;
                        break;
                    default:
                        break;
                    }
                }
            }
        }
    }
    return result; // retourne faux si l'Ã©vÃ©nement ferme l'application ou si l
a fenÃªtre est fermÃ©e
}

//======================================
// Accesseurs en Ã©criture
//======================================
void View::setModel(Model * model) {
    _model = model;
}
```