```
#include "Model.h"
#include "View.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <ctime>
#include <math.h>
#include "Obstacles.h"
#include "Score.h"
#include "AnimatedGraphicElement.h"
#include "Bonus.h"
using namespace std;

//=======================================
// Constructeurs
//=======================================
Model::Model(int w, int h):
    _w(w), _h(h) {
    _ball = new Ball(50, h-100, 30, 30, 0, 0);
    _score = new Score();
}

//=======================================
// Destructeurs
//=======================================
Model::~Model(){
    delete _score;
}

//=======================================
// Calcul la prochaine Ã©tape
//=======================================
void Model::nextStep(){

    time = clock.getElapsedTime();

    int C = rand()%3;
    if(time.asMilliseconds()>_clockDelay)     //boucle qui permet de determiner
le type de l'obstacle.
    {
        switch (C) {
        case 0:
            addElement("obstacle");
            break;
        case 1:
            addElement("obstacle2");
            break;
        case 2:
            addElement("obstacle3");
            break;
        default:
            break;
        }

        _clockDelay = 1000 + rand()%val;
        clock.restart();

        if(val != 0)
            val -= 10;
    }

    timeB = clockbonus.getElapsedTime();

    int B = rand()%2;
    if(timeB.asMilliseconds()>_clockBonus)   //boucle qui permet de determiner l
e type du bonus.
    {
        switch (B) {
        case 0:
```



```
            addElement("bonusvie");
            break;
        case 1:
            addElement("piece");
            break;
        default:
            break;
        }

        _clockBonus = 5000 + rand()%bonusshow;
        clockbonus.restart();
    }

    for(auto elm :_elements)
    {
        elm->move();
        if(elm->getX()+elm->getW() < 0) {
            _elementsTruck.push_back(elm);   //Si l'element est hors de la fenÃª
tre, on supprime l'Ã©lÃ©ment.
        }
        else if(collision(_ball, elm) == true)     //S'il y a collision avec la
balle, on place l'element dans _elementsTruck (qui sert
        {                                                                   //po
ur supprimer), et on retourne le rÃ©sultat de la collision (qui gÃ¨re le score e
t la barre de vie), si c'est soit un obstacle, soit un bonus.
            _elementsTruck.push_back(elm);
            resultatCollision(elm, C);
            resultatCollisionBonus(elm, B);
        }

    }

    for(auto it: _elementsTruck)        //Pour tous les Ã©lÃ©ments dans _element
sTruck, on les supprime
    {
        delete it;
        _elements.erase(find(_elements.begin(),_elements.end(),it));
    }
    _elementsTruck.clear();

    _ball->setdY(_ball->getdY()+_ball->g); // saut de la balle
    _ball->move();

    if (_ball->getY() > 450)
    {
        _ball->setY(450);
    }
}


//Fonction qui permet d'avoir la position de la balle.
void Model::getBallPosition(int&x, int&y)
{
    x = _ball->getX();
    y = _ball->getY();

}


//Fonction qui permet d'avoir la valeur x de la balle.
void Model::getXB(int &x)
{
    x = _ball->getX();

}


//Fonction qui permet d'avoir la valeur y de la balle.
void Model::getYB(int &y)
```

```
{
    y = _ball->getY();

}


//Fonction qui permet d'avoir la dimension de la balle.
void Model::getBallDimension(int&w, int&h)
{
    w = _ball->getW();
    h = _ball->getH();
}


//Fonction qui permet de dÃ©placer la balle.
void Model::moveBall(bool left)
{
    if(left)    //si left = true, on met la valeur de la vitesse de dÃ©placement
 Ã  -5
    {
        _ball->setdX(-5);
        _ball->setdY(0);
    }
    else     //sinon Ã  5
    {
        _ball->setdX(5);
        _ball->setdY(0);
    }
    _ball->move();
}


//Retourne le vecteur de MovableElement _new_elements
std::vector < const MovableElement * > Model::getNewMovableElements() const{
    return _new_elements;
}


//Retourne le vecteur de MovableElement _elements.
std::vector<MovableElement *> Model::getMovableElements() const {
    return _elements;
}


//Fonction qui permet d'ajouter des Ã©lÃ©ments suivant leur type.
void Model::addElement(std::string type)
{
    if (type=="obstacle")  //si c'est un obstacle simple, on crÃ©e un nouvel obs
tacle, et on le met dans le vecteur de MovableElement _new_elements.
    {
        Obstacles *ob= new Obstacles(SCREEN_WIDTH,450,100,50,0,-4);
        _new_elements.push_back(ob);
        _elements.push_back(ob);

    }
    if(type=="obstacle2")  //si c'est un obstacle moyen, on crÃ©er un nouvel obs
tacle, et on le met dans le vecteur de MovableElement _new_elements.
    {
        Obstacles *ob1= new Obstacles(SCREEN_WIDTH,400,100,100,1,-4);
        _new_elements.push_back(ob1);
        _elements.push_back(ob1);

    }
    if(type=="obstacle3")  //si c'est un obstacle dur, on crÃ©er un nouvel obsta
cle, et on le met dans le vecteur de MovableElement _new_elements.
    {
        Obstacles *ob2= new Obstacles(SCREEN_WIDTH,300,50,200,2,-4);
        _new_elements.push_back(ob2);
        _elements.push_back(ob2);
```



```
    }

    if(type=="piece"){  //si c'est une piÃ¨ce, on crÃ©er un nouveau bonus, et on
 le met dans le vecteur de MovableElement _new_elements.
        int w,h;
        getBallDimension(w,h);
        Bonus *bonus = new Bonus(SCREEN_WIDTH, 200, w, h, 4, -4);
        _new_elements.push_back(bonus);
        _elements.push_back(bonus);

    }

    if(type=="bonusvie"){   //si c'est une bonus de vie, on crÃ©er un nouveau bo
nus, et on le met dans le vecteur de MovableElement _new_elements.
        int w,h;
        getBallDimension(w,h);
        Bonus *bonusvie = new Bonus(SCREEN_WIDTH, 200, w, h, 5, -4);
        _new_elements.push_back(bonusvie);
        _elements.push_back(bonusvie);
    }

}


//Fonction qui permet de faire sauter la balle.
void Model::jumpBall()
{
    _ball->setdY(-jumpSpeed);
}


//Fonction qui retourne vrai si les coordonnÃ©es sont Ã©gales (<=> si il y a col
lision)
bool Model::collision(MovableElement *ball, MovableElement *obstacle)
{
    return ( ball->getX()+ball->getW() > obstacle->getX()
             && ball->getY()+ball->getH() > obstacle->getY()
             && ball->getX() < obstacle->getX()+obstacle->getW()
             && ball->getY() < obstacle->getY()+obstacle->getH()
             );
}


//Fonction qui permet de changer la position de l'obstacle.
void Model::setPositionObstacle(int x, int y)
{
    _obstacle->setPosition(x, y);
}


//Fonction qui retourne le rÃ©sultat d'une collision avec un obstacle.
void Model::resultatCollision(MovableElement * obstacle, int type)
{
    type = obstacle->getType();
    if(type == 0)  //Si c'est un obstacle simple, on perd 10pv (point de vie) et
 50 points de score.
    {
        _score->setVie(_score->getVie()-10);
        _score->setScore(_score->getScore()-50);
    }
    else if(type == 1)  //Si c'est un obstacle moyen, on perd 15pv et 100 points
 de score.
    {
        _score->setVie(_score->getVie()-15);
        _score->setScore(_score->getScore()-100);
    }
    else if(type == 2)  //Si c'est un obstacle dur, on perd 20pv et 200 points d
e score.
```

```
    {
        _score->setVie(_score->getVie()-20);
        _score->setScore(_score->getScore()-200);
    }

}


//Fonction qui retourne le rÃ©sultat d'une collision avec un bonus.
void Model::resultatCollisionBonus(MovableElement *bonus, int type)
{
    type = bonus->getType();
    if(type == 4) //Si c'est une piÃ¨ce, on gagne 100 points de score
    {
        _score->setScore(_score->getScore()+100);
    }
    else if (type == 5 && _score->getVie() <= 100)  //Si c'est un bonus de vie,
on gagne 50pv.
    {
        _score->setVie(_score->getVie()+50);
    }
}


//Fonction qui permet d'Ã©crire le score (en string)
string Model::writeScore(int score){
    ostringstream ss ;
    ss << score;
    return ss.str();

}


//Fonction qui retourne le score
int Model::getScore(){
    return _score->getScore();
}


//Fonction qui permet de changer le score
void Model::setScore(int s){
    _score->setScore(s);
}


//Fonction qui retourne la vie
int Model::getVie(){
    return _score->getVie();
}

//Fonction qui permet de changer la vie
void Model::setVie(int nb){
    _score->setVie(nb);
}

//Fonction qui appelle les fonctions de gestion des meilleurs scores (qui sont i
mplentÃ©es mais pas opÃ©rationnelles).
void Model::bestScores(){
    _score->initHScores();
    _score->addHScore();
    _score->SaveHScore();
}
```