```
#include "Camion.h"

using namespace std;

//Contructeur
Camion::Camion(): Vehicule(), m_vitesse(1)
{}

//Destructeur
Camion::~Camion()
{}

//Accesseur en lecture
int Camion::getVitesse() const
{
	return m_vitesse;
}
```

```
#ifndef _CAMION_H_
#define _CAMION_H_

#include "Vehicule.h"

class Camion: public Vehicule
{
        private :
                int m_vitesse;
        public :
                Camion();
                ~Camion();
                int getVitesse() const;
};

#endif
```

```
#ifndef _CONSTANTES_H_
#define _CONSTANTES_H_

const int LARGEUR_FENETRE=800;
const int HAUTEUR_FENETRE=600;
const int TAILLE_LARGEUR_BOUTON = 250;
const int TAILLE_HAUTEUR_BOUTON = 75;
const int NB_ROUTES_MAX=6;
const int TAILLE_ROUTE=40;
const int TAILLE_LARGEUR_FEU = 25;
const int TAILLE_HAUTEUR_FEU = 49;
const int TAILLE_BARRE_PARTIE = 40;
const int ECART_HORIZONTAL_ROUTES=(LARGEUR_FENETRE-TAILLE_ROUTE*NB_ROUTES_MAX/2)/(NB_ROUTES_MAX/2+1);
const int ECART_VERTICAL_ROUTES=(HAUTEUR_FENETRE-TAILLE_ROUTE*NB_ROUTES_MAX/2)/(NB_ROUTES_MAX/2+1);
const int TAILLE_LONGUEUR_VOITURE_ET_POLICE=66;
const int TAILLE_LARGEUR_VOITURE_ET_POLICE=30;
const int TAILLE_LONGUEUR_CAMION=80;
const int TAILLE_LARGEUR_CAMION=30;
const float BASE_VITESSE = 1;

#endif
```

```
#include "Croisement.h"

using namespace std;

//Constructeur
Croisement::Croisement() : Objet(), m_feu(0), m_presenceFeu(0), m_nbVehiculePresent(0), m_collision(0)
{}

//Destructeur
Croisement::~Croisement()
{
        if (m_presenceFeu)
                delete m_feu;
}

//Accesseurs en lecture
Feu* Croisement::getFeu() const
{
        return m_feu;
}

bool Croisement::getPresenceFeu() const
{
        return m_presenceFeu;
}

int Croisement::getNbVehiculePresent() const
{
        return m_nbVehiculePresent;
}

//Accesseurs en Ã©criture
void Croisement:: setPresenceFeu(bool presenceFeu)
{
        m_presenceFeu=presenceFeu;
}

void Croisement::setPosXCroisement(vector <Croisement*> listeDeCroisement, int posX, int i)/**/
{
        listeDeCroisement[i]->setPosX(posX);
}

void Croisement::setNbVehiculePresent(int nbVehicule)
{
        m_nbVehiculePresent = nbVehicule;
}

//MÃ©thodes
void Croisement::genererFeu()
{
        m_feu=new Feu();
}

bool Croisement::detecterCollision()
{
        if (m_nbVehiculePresent > 1)
        {
                m_collision = 1;
        }
        return m_collision;
}
```

```
#ifndef _CROISEMENT_H_
#define _CROISEMENT_H_

#include "Objet.h"
#include "Feu.h"
#include <vector>

class Croisement : public Objet
{
        private :
                Feu* m_feu;
                bool m_presenceFeu;
                int m_nbVehiculePresent;
                bool m_collision;
        public :
                Croisement();
                ~Croisement();
                bool getPresenceFeu() const;
                Feu* getFeu() const;
                int getNbVehiculePresent() const;
                void setPresenceFeu(bool presenceFeu);
                void setPosXCroisement(std::vector <Croisement*> listeDeCroisement, int posX, int i);
                void setNbVehiculePresent(int nbVehicule);
                void genererFeu();
                bool detecterCollision();
};

#endif
```

```
#include "Feu.h"

using namespace std;

//Constructeur
Feu::Feu(): m_estVert(0)
{}

//Destructeur
Feu::~Feu()
{}

//Accesseur en lecture
bool Feu::getCouleurFeu() const
{
        return m_estVert;
}

//Accesseur en Ã©criture
void Feu::changerCouleurFeu()
{
        switch (m_estVert)
        {
                case 0: m_estVert=1; break;
                case 1: m_estVert=0; break;
        }
}
```

```
#ifndef _FEU_H_
#define _FEU_H_

#include "Objet.h"

class Feu: public Objet
{
        private:
                bool m_estVert;
        public:
                Feu();
                ~Feu();
                bool getCouleurFeu() const;
                void changerCouleurFeu();
};

#endif
```

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

using namespace std;
using namespace sf;

// Constructeurs
GameModel::GameModel()
: m_width(LARGEUR_FENETRE), m_height(HAUTEUR_FENETRE), m_nbVehicules_Horizontal(0), m_nbVehicules_Vertical(0), time_horizontal(0), time_vertical(0), m_xCollisio
n(0), m_yCollision(0), m_tempsCollision(0), m_nbAlea(Randomizer::Random(1,3))
{}

GameModel::GameModel(int width, int height)
:  m_width(width), m_height(height), m_nbVehicules_Horizontal(0), m_nbVehicules_Vertical(0), time_horizontal(0), time_vertical(0), m_xCollision(0), m_yCollision
(0), m_tempsCollision(0), m_nbAlea(Randomizer::Random(0,2))
{
        m_partie=new Partie();
        m_partie->genererNiveau();
}


// Destructeurs
GameModel::~GameModel()
{
        delete m_partie;
}

//Accesseur en lecture
Partie* GameModel ::getPartie() const
{
        return m_partie;
}

float GameModel::getTemps() const
{
        return horloge_collision.GetElapsedTime();
}

float GameModel:: getTempsCollision() const
{
        return m_tempsCollision;
}

int GameModel:: getXCollision() const
{
        return m_xCollision;
}

int GameModel:: getYCollision() const
{
        return m_yCollision;
}

// Calcul la prochaine Ã©tape
void GameModel::nextStep()
{
        collision();
        for (int i=0; i<getPartie()->getRoutes().size() ; i++)
        {
                        // Timer
                        time_horizontal = horloge_horizontal.GetElapsedTime();
                        time_vertical = horloge_vertical.GetElapsedTime();
                        if (m_nbVehicules_Horizontal <= getPartie()->getNiveauEnCours() && getPartie()->getRoutes()[i]->getNumeroVehicule() < getPartie()->getRo
utes()[i]->getListeDeVehicules().size())
                        {
                                if (time_horizontal > 2)
                                {
                                        if (!getPartie()->getRoutes()[i]->getSens())
                                        {
                                                for (int t=0 ; t < getPartie()->getRoutes().size() ; t++)
                                                {
                                                        if (!getPartie()->getRoutes()[t]->getSens())
                                                        {
                                                                getPartie()->getRoutes()[t]->getListeDeVehicules()[getPartie()->getRoutes()[t]->getNumeroVehicul
e()]->setEtatVehicule(1);

                                                                getPartie()->getRoutes()[t]->setNumeroVehicule(getPartie()->getRoutes()[t]->getNumeroVehicule()+
1);

                                                        }
                                                }
                                                m_nbVehicules_Horizontal++;
                                                horloge_horizontal.Reset();
                                                time_horizontal = 0;
                                                if (m_nbVehicules_Horizontal == getPartie()->getNiveauEnCours())
                                                {
                                                        m_nbVehicules_Horizontal = 0;
                                                }
                                        }
                                }
                                else if (time_vertical > m_nbAlea)
                                {
                                        for (int f=0 ; f < getPartie()->getRoutes().size(); f++)
                                        {
                                        if (getPartie()->getRoutes()[f]->getSens())
                                        {
                                                getPartie()->getRoutes()[f]->getListeDeVehicules()[getPartie()->getRoutes()[f]->getNumeroVehicule()]->setEtatVeh
icule(1);

                                                getPartie()->getRoutes()[f]->setNumeroVehicule(getPartie()->getRoutes()[f]->getNumeroVehicule()+1);
                                                m_nbVehicules_Vertical++;
                                                horloge_vertical.Reset();
                                                time_vertical = 0;
                                                m_nbAlea = Randomizer::Random(1,4);
                                                if (m_nbVehicules_Vertical == getPartie()->getNiveauEnCours()/2)
                                                {
                                                        m_nbVehicules_Vertical = 0;
                                                }
                                        }
                                        }
                                }
                                else
                                {
                                        for (int v=0 ; v < getPartie()->getRoutes()[i]->getListeDeVehicules().size() ; v++)
                                        {
                                                if (v >= getPartie()->getRoutes()[i]->getNumeroVehicule())
                                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[v]->setEtatVehicule(0);
                                        }
                                }
                        }
                // Avancement des vÃ©hicules
                for (int j=0 ; j<getPartie()->getRoutes()[i]->getListeDeVehicules().size() ; j++)
                {
                        if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getEtatVehicule())
                        {
                                if (getPartie()->getRoutes()[i]->getSens())
```



```
                                switch (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getVitesse())
                                {
                                        case 1 :
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosY
                                                (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()+BASE_VITESSE*1.5);
                                                break;
                                        }
                                        case 2 :
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosY
                                                (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()+BASE_VITESSE*2);
                                                break;
                                        }
                                        case 3 :
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosY
                                                (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()+BASE_VITESSE*3);
                                                break;
                                        }
                                }
                        else if (!getPartie()->getRoutes()[i]->getSens())
                                switch (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getVitesse())
                                {
                                        case 1 :
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosX
                                                (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()+BASE_VITESSE*1.5);
                                                break;
                                        }
                                        case 2 :
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosX
                                                (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()+BASE_VITESSE*2);
                                                break;
                                        }
                                        case 3 :
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosX
                                                (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()+BASE_VITESSE*3);
                                                break;
                                        }
                                }
                        }
                        // ArrÃªt des vÃ©hicules hors de la fenÃªtre
                        if (!getPartie()->getRoutes()[i]->getSens())
                        {
                                if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()==LARGEUR_FENETRE ||
                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()==LARGEUR_FENETRE+2)
                                {
                                        getPartie()->getNiveau()->setNbVehiculesPasses(getPartie()->getNiveau()->getNbVehiculesPasses()+1);
                                        getPartie()->setScore(getPartie()->calculerScore(i,j));
                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setPosX(1200);
                                        getPartie()->getNiveau()->setNbVehiculePresent(getPartie()->getNiveau()->getNbVehiculePresent()-1);
                                }

                                if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() == -2)
                                        getPartie()->getNiveau()->setNbVehiculePresent(getPartie()->getNiveau()->getNbVehiculePresent()+1);
                        }
                }
        }
        if (getPartie()->getNiveau()->getNbVehiculesPasses() == getPartie()->getNiveau()->getObjectif())
        {
                horloge_horizontal.Reset();
                horloge_vertical.Reset();
        }
}

void GameModel:: collision()
{
                float taille_vehicule;
                for (int i=0 ; i < getPartie()->getRoutes().size() ; i++)
                {
                        for (int j=0 ; j < getPartie()->getRoutes()[i]->getListeDeVehicules().size() ; j++)
                        {
                                switch (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getVitesse())
                                {
                                        case 1 : taille_vehicule = TAILLE_LONGUEUR_CAMION; break;
                                        case 2 : taille_vehicule = TAILLE_LONGUEUR_VOITURE_ET_POLICE; break;
                                        case 3 : taille_vehicule = TAILLE_LONGUEUR_VOITURE_ET_POLICE; break;
                                        default : break;
                                }
                                if (j>0 && getPartie()->getRoutes()[i]->getListeDeVehicules()[j-1] != NULL)
                                {
                                        if (!getPartie()->getRoutes()[i]->getSens())
                                        {
                                                if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() + taille_vehicule
                                                        >= getPartie()->getRoutes()[i]->getListeDeVehicules()[j-1]->getPositionX())
                                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(0);
                                                else
                                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(1);
                                        }
                                        else if (getPartie()->getRoutes()[i]->getSens())
                                        {
                                                if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY() + taille_vehicule
                                                        >= getPartie()->getRoutes()[i]->getListeDeVehicules()[j-1]->getPositionY())
                                                {
                                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(0);
                                                }
                                                else
                                                {
                                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(1);
                                                }
                                        }
                                }
                                else if (j==0)
                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(1);
                                else if (j>0 && getPartie()->getRoutes()[i]->getListeDeVehicules()[j-1] == NULL)
                                        getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(1);
                                // Gestion de l'arrÃªt des vÃ©hicules derriÃ¨re celui le prÃ©cÃ©dant
                                if (!getPartie()->getRoutes()[i]->getSens())
                                {
                                        for (int k=0 ; k<getPartie()->getRoutes()[i]->getListeDeCroisements().size() ; k++)
                                        {
                                                if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() + taille_vehicule
                                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX()
                                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() + taille_vehicule
                                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX()- 10
                                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()
                                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY()
                                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()
                                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY() + 40
                                                        && getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPresenceFeu()
                                                        && !getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getFeu()->getCouleurFeu())
                                                        {
```

```
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(0);
                                        }

                                else if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() + taille_vehicule
                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX()
                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() + taille_vehicule
                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX()- 10
                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()
                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY()
                                        &&getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()
                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY() + 40
                                        && getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPresenceFeu()
                                        && getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getFeu()->getCouleurFeu())
                                        {
                                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(1);
                                        }

                                // Gestion des collisions sur les croisements
                                else if (getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX() + taille_vehicule
                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX()
                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()
                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX() - taille_vehicule
                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionX()
                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX() + TAILLE_ROUTE
                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()
                                        > getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY()
                                        && getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY()
                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY() + TAILLE_ROUTE)
                                {
                                        if (getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getNbVehiculePresent() == 1)
                                        {
                                                for (int l=0 ; l<getPartie()->getRoutes().size() ; l++)
                                                        if(getPartie()->getRoutes()[l]->getSens())
                                                        for (int v=0 ; v < getPartie()->getRoutes()[l]->getListeDeVehicules().size() ; v++)
                                                                if (getPartie()->getRoutes()[l]->getListeDeVehicules()[v]->getPositionY() + tail
le_vehicule
                                                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY
()
                                                                        && getPartie()->getRoutes()[l]->getListeDeVehicules()[v]->getPositionY()
                                                                        >= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY
() - taille_vehicule
                                                                        && getPartie()->getRoutes()[l]->getListeDeVehicules()[v]->getPositionY()
                                                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionY
() + TAILLE_ROUTE
                                                                        && getPartie()->getRoutes()[l]->getListeDeVehicules()[v]->getPositionX()
                                                                        > getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX(
)
                                                                        && getPartie()->getRoutes()[l]->getListeDeVehicules()[v]->getPositionX()
                                                                        <= getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->getPositionX
() + TAILLE_ROUTE)
                                                                {
                                                                        if (getTemps() > 2)
                                                                                horloge_collision.Reset();
                                                                        m_xCollision = getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->
getPositionX();
                                                                        m_yCollision = getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->
getPositionY();

                                                                        m_tempsCollision = getTemps();
                                                                        getPartie()->getNiveau()->setCollision(1);
                                                                        horloge_horizontal.Reset();
                                                                        horloge_vertical.Reset();
                                                                }
                                        }
                                        else
                                                getPartie()->getRoutes()[i]->getListeDeCroisements()[k]->setNbVehiculePresent(1);
                                }
                        }
                }
        }

        //ArrÃªt des vÃ©hicules lors d'une collision
        if (getPartie()->getNiveau()->getCollision())
        {
                for (int i=0; i<getPartie()->getRoutes().size(); i++)
                        for (int j=0; j<getPartie()->getRoutes()[i]->getListeDeVehicules().size(); j++)
                        {
                                getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->setEtatVehicule(0);
                        }
        }
}
```

```
#ifndef _GAME_MODEL_H_
#define _GAME_MODEL_H_

#include <SFML/System.hpp>
#include "Partie.h"

class GameModel {
private:
        int m_width, m_height;
        int m_nbVehicules_Horizontal;
        int m_nbVehicules_Vertical;
        int m_xCollision, m_yCollision;
        float m_tempsCollision;
        bool m_collisionFinie;

        sf::Clock horloge_horizontal;
        sf::Clock horloge_vertical;
        sf::Clock horloge_collision;
        float time_horizontal;
        float time_vertical;
        int m_nbAlea;
        Partie* m_partie;


public:

        GameModel();
        GameModel(int width, int height);
        ~GameModel();
        Partie* getPartie() const;
        float getTemps() const;
        float getTempsCollision() const;
        int getXCollision() const;
        int getYCollision() const;
        void nextStep();
        void collision();


};
#endif
```

```
#include <cstdlib>
#include <iostream>
#include <sstream>
#include "GameView.h"
#include "GameModel.h"
#include "Niveau.h"
#include "Constantes.h"
#include "Partie.h"

using namespace std;
using namespace sf;


// Constructeurs

GameView::GameView(int width, int height): m_width(width), m_height(height), m_menu(true), m_animation(true)
{
	m_window = new RenderWindow(sf::VideoMode(width, height, 32), "TrafficJam", sf::Style::Close);
	m_chargement.SetFromRect(sf::FloatRect(0, 0, width, height));
	m_interNiveau.SetFromRect(sf::FloatRect(0,0, width, height));

	if (!m_font.LoadFromFile("images/AlteHaasGroteskBold.ttf", 50))
	{
		cout << "Erreur lors du chargement de la police"<<endl;
	}
	else if (!m_image_logo_iut.LoadFromFile("images/logo.jpeg") ||
		 !m_image_bouton_jouer.LoadFromFile("images/bouton_jouer.png") ||
		 !m_image_bouton_quitter.LoadFromFile("images/bouton_quitter.png") ||
		 !m_image_h_police.LoadFromFile("images/hpolice.png") ||
		 !m_image_background.LoadFromFile("images/grass2.jpg") ||
		 !m_image_vRoute.LoadFromFile("images/vroad.jpg") ||
		 !m_image_hRoute.LoadFromFile("images/hroad.jpg") ||
		 !m_image_h_voiture.LoadFromFile("images/hcar.png") ||
		 !m_image_h_camion.LoadFromFile("images/hbus.png") ||
		 !m_image_v_police.LoadFromFile("images/vpolice.png") ||
		 !m_image_v_voiture.LoadFromFile("images/vcar.png") ||
		 !m_image_v_camion.LoadFromFile("images/vbus.png") ||
		 !m_image_croisement.LoadFromFile("images/crossroad.jpg") ||
		 !m_image_feu_rouge.LoadFromFile("images/redlight.jpg") ||
		 !m_image_feu_vert.LoadFromFile("images/greenlight.jpg") ||
		 !m_image_ligne.LoadFromFile("images/lightline.jpg") ||
		 !m_image_explosion.LoadFromFile("images/explosion.png"))
	{
		cout << "Erreur lors du chargement du menu" <<endl;
	}
}


// Destructeur
GameView::~GameView()
{
	if(m_window!= NULL)
		delete m_window;
}

// Accesseurs en Ã©criture
void GameView::setModel(GameModel * model)
{
	m_model = model;
}

bool GameView::getMenu(GameView * view) const
{
	return m_menu;
}

// Fonction de dessin
string GameView::convertInt(int number)
{
	stringstream ss;
	ss << number;
	return ss.str();
}

void GameView::ecranChargement()
{
	m_window->Clear(sf::Color(37,38,35));
	m_window->Draw (m_intro);
	m_window->Draw (m_sprite_logo_iut);

	m_window->SetView(m_chargement);
	m_window->Display();
}

void GameView::ecranInterNiveau()
{
	Event event;
	horloge.Reset();
	while ((m_window->GetEvent(event) || !m_menu) && horloge.GetElapsedTime() < 5.f)
	{
			m_window->Clear(sf::Color(37,38,35));
			if (m_model->getPartie()->getNiveau()->getNbVehiculesPasses() == m_model->getPartie()->getNiveau()->getObjectif())
			{
				m_texte_gagne = "   Niveau Suivant \n\n       " + convertInt(5-horloge.GetElapsedTime());
				m_texte_interNiveau_gagne.SetText(m_texte_gagne);
				m_window->Draw(m_texte_interNiveau_gagne);
			}
			else if (m_model->getPartie()->getNiveau()->getNbVehiculePresent() == m_model->getPartie()->getNiveau()->getTrafficAutorise()
					|| m_model->getPartie()->getNiveau()->getCollision())
			{
				m_texte_perdu = "      Perdu \n\n       " + convertInt(5-horloge.GetElapsedTime());
				m_texte_interNiveau_perdu.SetText(m_texte_perdu);
				m_window->Draw(m_texte_interNiveau_perdu);
			}

			sf::String m_texte_retour_menu("Appuyez sur Espace pour revenir au Menu", m_font, 35);
			m_texte_retour_menu.SetPosition(50, 400);
			m_texte_retour_menu.SetColor(sf::Color(255,204,102));
			m_window->Draw (m_texte_retour_menu);

			if ((event.Type == sf::Event::KeyPressed) && (event.Key.Code == sf::Key::Space))
			{
				m_menu = true;
			}
			m_window->SetView(m_interNiveau);
			m_window->Display();


			m_animation = false;
	}
}

void GameView::declarationImages()
{
	if (m_menu)
```



```
	{
		m_intro.SetText(L"      TrafficJam \n Par GaÃ«tan Roudeau \n et Anthony Silverio");
		m_intro.SetFont(m_font);
		m_intro.SetSize(60);
		m_intro.SetColor(sf::Color(255,204,102));
		m_intro.SetPosition(100, 100);

		m_sprite_logo_iut = Sprite (m_image_logo_iut);
		m_sprite_logo_iut.Resize(125, 100);
		m_sprite_logo_iut.SetPosition(350, 400);

		m_sprite_h_police = Sprite (m_image_h_police);
		m_sprite_h_police.Rotate(45);
		m_sprite_h_police.Resize(300,120);
		m_sprite_h_police.SetPosition(100,400);

		m_titre.SetText(L"TrafficJam");
		m_titre.SetFont(m_font);
		m_titre.SetSize(50);
		m_titre.SetColor(sf::Color(255,204,102));
		m_titre.SetPosition(275,50);
		m_titre.SetStyle(sf::String::Underlined);

		m_sprite_bouton_jouer = Sprite (m_image_bouton_jouer);
		m_sprite_bouton_jouer.Resize(TAILLE_LARGEUR_BOUTON , TAILLE_HAUTEUR_BOUTON);
		m_sprite_bouton_jouer.SetPosition(500,225);
		m_sprite_bouton_quitter = Sprite (m_image_bouton_quitter);
		m_sprite_bouton_quitter.Resize(TAILLE_LARGEUR_BOUTON , TAILLE_HAUTEUR_BOUTON);
		m_sprite_bouton_quitter.SetPosition(500,425);
	}
	else if (!m_menu && !m_animation)
	{
		// Ecran Inter-Niveau
		m_texte_interNiveau_gagne.SetFont(m_font);
		m_texte_interNiveau_gagne.SetSize(60);
		m_texte_interNiveau_gagne.SetColor(sf::Color(255,204,102));
		m_texte_interNiveau_gagne.SetPosition(100, 100);

		m_texte_interNiveau_perdu.SetFont(m_font);
		m_texte_interNiveau_perdu.SetSize(60);
		m_texte_interNiveau_perdu.SetColor(sf::Color(255,204,102));
		m_texte_interNiveau_perdu.SetPosition(100, 100);

		// Ecran de jeux regroupant le fond, les routes, les croisements, les voitures
		m_sprite_background = Sprite (m_image_background);
		m_sprite_background.Resize(m_width, m_height);
		m_sprite_background.SetPosition(0,0);

		m_sprite_vRoute = Sprite (m_image_vRoute);
		m_sprite_vRoute.Resize(TAILLE_ROUTE, m_height - TAILLE_BARRE_PARTIE);

		m_sprite_hRoute = Sprite (m_image_hRoute);
		m_sprite_hRoute.Resize(m_width, TAILLE_ROUTE);

		m_sprite_croisement = Sprite (m_image_croisement);

		m_sprite_h_voiture = Sprite (m_image_h_voiture);
		m_sprite_h_voiture.Resize(TAILLE_LONGUEUR_VOITURE_ET_POLICE, TAILLE_LARGEUR_VOITURE_ET_POLICE);
		m_sprite_v_voiture = Sprite (m_image_v_voiture);
		m_sprite_v_voiture.Resize(TAILLE_LARGEUR_VOITURE_ET_POLICE, TAILLE_LONGUEUR_VOITURE_ET_POLICE);

		m_sprite_h_camion = Sprite (m_image_h_camion);
		m_sprite_h_camion.Resize(TAILLE_LONGUEUR_CAMION, TAILLE_LARGEUR_CAMION);
		m_sprite_v_camion = Sprite (m_image_v_camion);
		m_sprite_v_camion.Resize(TAILLE_LARGEUR_CAMION, TAILLE_LONGUEUR_CAMION);

		m_sprite_h_police = Sprite (m_image_h_police);
		m_sprite_h_police.Resize(TAILLE_LONGUEUR_VOITURE_ET_POLICE,TAILLE_LARGEUR_VOITURE_ET_POLICE);
		m_sprite_v_police = Sprite (m_image_v_police);
		m_sprite_v_police.Resize(TAILLE_LARGEUR_VOITURE_ET_POLICE,TAILLE_LONGUEUR_VOITURE_ET_POLICE);

		// Barre partie regroupant les infos
		m_barre_partie = sf::Shape::Rectangle(0, 560, 800, 600, sf::Color(0,0,0));

		int niveauCours = m_model->getPartie()->getNiveauEnCours();
		m_niveau = "Niveau : " + convertInt(niveauCours);
		m_niveauEnCours = sf::String (m_niveau, m_font, 25);
		m_niveauEnCours.SetPosition(10,562);

		int objectifVoitures = m_model->getPartie()->getNiveau()->getObjectif();
		int nbVehiculesAtteint = m_model->getPartie()->getNiveau()->getNbVehiculesPasses();
		m_objectif = "Vehicules " + convertInt(nbVehiculesAtteint) + "/" + convertInt(objectifVoitures);
		m_nbVehiculesObjectif = sf::String (m_objectif, m_font, 25);
		m_nbVehiculesObjectif.SetPosition(140, 562);

		m_traffic = sf::String ("Traffic", m_font, 25);
		m_traffic.SetPosition(320, 562);
		m_barre_traffic = sf::Shape::Rectangle(410, 566, 590, 593, sf::Color::Black, 3, sf::Color::White);
		float traffic_actuel = 410 + (180 * m_model->getPartie()->getNiveau()->getNbVehiculePresent() / m_model->getPartie()->getNiveau()->getTrafficAut
orise());
		m_barre_traffic_actuel = sf::Shape::Rectangle(410, 566, traffic_actuel, 593, sf::Color::Green);
		if (traffic_actuel >= 469 && traffic_actuel < 528)
			m_barre_traffic_actuel = sf::Shape::Rectangle(410, 566, traffic_actuel, 593, sf::Color(205, 102, 0));
		else if (traffic_actuel >= 528)
			m_barre_traffic_actuel = sf::Shape::Rectangle(410, 566, traffic_actuel, 593, sf::Color::Red);

		int scoreActu = m_model->getPartie()->getScore();
		m_score = "Score : " + convertInt(scoreActu) + " pts" ;
		m_scoreActuel = sf::String (m_score, m_font, 25);
		m_scoreActuel.SetPosition(620 ,562);
	}
}

void GameView::draw()
{
	if (m_menu && m_animation)
	{
		while (horloge.GetElapsedTime() < 2.f)
			ecranChargement();
		m_animation = false;
		m_window->SetView(m_window->GetDefaultView());
	}

	else if (m_menu && !m_animation)
	{
		m_window->Clear(sf::Color(37,38,35));
		m_window->Draw (m_sprite_h_police);
		m_window->Draw (m_titre);
		m_window->Draw (m_sprite_bouton_jouer);
		m_window->Draw (m_sprite_bouton_quitter);
		m_window->Display();
	}

	else if (!m_menu && !m_animation)
	{
```

```
                m_window->Draw (m_sprite_background);
                this->affichageEcran();
                m_window->Draw(m_barre_partie);
                m_window->Draw(m_niveauEnCours);
                m_window->Draw(m_nbVehiculesObjectif);
                m_window->Draw(m_traffic);
                m_window->Draw(m_barre_traffic);
                m_window->Draw(m_barre_traffic_actuel);
                m_window->Draw(m_scoreActuel);
                if (m_model->getPartie()->getNiveau()->getNbVehiculePresent() == m_model->getPartie()->getNiveau()->getTrafficAutorise()
                        || m_model->getPartie()->getNiveau()->getNbVehiculesPasses() == m_model->getPartie()->getNiveau()->getObjectif())
                {
                        m_animation = true;
                }
                m_window->Display();
        }

        else if (!m_menu && m_animation)
        {
                ecranInterNiveau();
                m_window->SetView(m_window->GetDefaultView());
                if (m_model->getPartie()->getNiveau()->getNbVehiculePresent() == m_model->getPartie()->getNiveau()->getTrafficAutorise()
                        || m_model->getPartie()->getNiveau()->getCollision())
                {
                        m_model->getPartie()->supprimerNiveau();
                        m_model->getPartie()->setNiveauEnCours(m_model->getPartie()->getNiveauEnCours());
                        m_model->getPartie()->genererNiveau();
                }
                else if (m_model->getPartie()->getNiveau()->getNbVehiculesPasses() == m_model->getPartie()->getNiveau()->getObjectif())
                {
                        m_model->getPartie()->supprimerNiveau();
                        m_model->getPartie()->setNiveauEnCours(m_model->getPartie()->getNiveauEnCours()+1);
                        m_model->getPartie()->genererNiveau();
                }
        }
        usleep(500);
}

void GameView::affichageEcran()
{
                for (int i=0; i<m_model->getPartie()->getRoutes().size();i++)
                        {
                                if (m_model->getPartie()->getRoutes()[i]->getSens())
                                {
                                        m_sprite_vRoute.SetPosition(m_model->getPartie()->getRoutes()[i]->getPositionX(),0);
                                        m_window->Draw (m_sprite_vRoute);

                                }
                                else
                                {
                                        m_sprite_hRoute.SetPosition(0,m_model->getPartie()->getRoutes()[i]->getPositionY());
                                        m_window->Draw (m_sprite_hRoute);

                                }
                        }
                for (int i=0; i<m_model->getPartie()->getRoutes().size();i++)
                {
                        affichageCroisement(i);
                }
                for (int i=0; i<m_model->getPartie()->getRoutes().size() ; i++)
                {
                        for (int j=0 ; j<m_model->getPartie()->getRoutes()[i]->getListeDeVehicules().size() ; j++)
                        {
                                switch (m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getVitesse())
                                {
                                        case 1 :
                                        {
                                                if (!m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getSens())
                                                {
                                                        m_sprite_h_camion.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositio
nX(),
                                                                                      m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY());
                                                        m_window->Draw(m_sprite_h_camion);
                                                }
                                                else if (m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getSens())
                                                {
                                                        m_sprite_v_camion.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositio
nX(),
                                                                                      m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY());
                                                        m_window->Draw(m_sprite_v_camion);
                                                }
                                                break;
                                        }
                                        case 2 :
                                        {
                                                if (!m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getSens())
                                                {
                                                        m_sprite_h_voiture.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositi
onX(),
                                                                                       m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY());
                                                        m_window->Draw(m_sprite_h_voiture);
                                                }
                                                else if (m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getSens())
                                                {
                                                        m_sprite_v_voiture.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositi
onX(),
                                                                                       m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY());
                                                        m_window->Draw(m_sprite_v_voiture);
                                                }
                                                break;
                                        }
                                        case 3 :
                                        {
                                                if (!m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getSens())
                                                {
                                                        m_sprite_h_police.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositio
nX(),
                                                                                      m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY());
                                                        m_window->Draw(m_sprite_h_police);
                                                }
                                                else if (m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getSens())
                                                {
                                                        m_sprite_v_police.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositio
nX(),
                                                                                      m_model->getPartie()->getRoutes()[i]->getListeDeVehicules()[j]->getPositionY());
                                                        m_window->Draw(m_sprite_v_police);
                                                }
                                                break;
                                        }
                                }
                        }
                }
                //affichage collision
                if (m_model->getPartie()->getNiveau()->getCollision())
                {
```



```
                        m_sprite_explosion = Sprite (m_image_explosion);
                        float dureeCollision = m_model->getTempsCollision();
                        if(dureeCollision<=0.05)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(0,0,65,65));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.1)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(65,0,130,65));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.15)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(130,0,195,65));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.2)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(195,0,256,65));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.25)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(0,65,65,130));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.3)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(65,65,130,130));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.35)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(130,65,195,130));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.40)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(195,65,256,130));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.45)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(0,130,65,195));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.5)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(65,130,130,195));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.55)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(130,130,195,195));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.6)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(195,130,256,195));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.65)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(0,195,65,256));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.7)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(65,195,130,256));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.75)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(130,195,195,256));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        else if (dureeCollision<=0.8)
                        {
                                m_sprite_explosion.SetSubRect(sf::IntRect(195,195,256,256));
                                m_sprite_explosion.SetPosition(m_model->getXCollision()-10, m_model->getYCollision()-10);
                                m_window->Draw (m_sprite_explosion);
                        }
                        if (dureeCollision >= 0.9)
                        {
                                m_animation = true;
                        }
                }
}

void GameView:: affichageCroisement(int i)
{
        if (!m_model->getPartie()->getRoutes()[i]->getSens())
                                for (int j=0; j < m_model->getPartie()->getRoutes()[i]->getListeDeCroisements().size() ; j++)
                                        {
                                                m_sprite_croisement.SetPosition(m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPositionX()
,
                                                                                m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPositionY()
);
                                                m_window->Draw (m_sprite_croisement);
                                                if (m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPresenceFeu())
                                                {
                                                        switch (m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getFeu()->getCouleurFeu())
                                                        {
                                                                case 0 :
                                                                        {
                                                                                m_sprite_feu = Sprite (m_image_feu_rouge);
                                                                                m_sprite_feu.Resize(TAILLE_LARGEUR_FEU, TAILLE_HAUTEUR_FEU);
                                                                                break;
                                                                        }
```

```
                                                        case 1 :
                                                                {
                                                                        m_sprite_feu = Sprite (m_image_feu_vert);
                                                                        m_sprite_feu.Resize(TAILLE_LARGEUR_FEU, TAILLE_HAUTEUR_FEU);
                                                                        break;
                                                                }
                                                }
                                                m_sprite_feu.SetPosition
                                                (m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPositionX() - TAILLE_LARGEUR_FEU,
                                                m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPositionY() - TAILLE_HAUTEUR_FEU);
                                                m_window->Draw(m_sprite_feu);

                                                m_sprite_ligne = Sprite (m_image_ligne);
                                                m_sprite_ligne.Rotate(270);
                                                m_sprite_ligne.Resize(40, 10);
                                                m_sprite_ligne.SetPosition
                                                        (m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPositionX(),
                                                m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPositionY());
                                                m_window->Draw(m_sprite_ligne);

                                        }
                                }
}
// Traitement des evenements
bool GameView::treatEvents(){
        bool result = false;
        const sf::Input& input = m_window->GetInput();
        this->declarationImages();

        if(m_window->IsOpened()){
                result = true;
                sf::Event event;
                while (m_window->GetEvent(event))
                {
                        this->declarationImages();
                        int mouse_x = input.GetMouseX();
                        int mouse_y = input.GetMouseY();
                        if ((event.Type == sf::Event::Closed) ||
                                ((event.Type == sf::Event::KeyPressed) && (event.Key.Code == sf::Key::Escape)))
                        {
                                m_window->Close();
                                result = false;
                        }
                        if (m_menu)
                        {
                                // Gestion des clics sur les boutons du menu et de la position de la souris sur ceux-ci
                                if((event.Type == sf::Event::MouseButtonPressed) && (event.MouseButton.Button == sf::Mouse::Left))
                                {
                                        if (mouse_x >= 500 && mouse_x <= 750 && mouse_y >= 225 && mouse_y <= 300)
                                        {
                                                m_menu=false;
                                                this->declarationImages();
                                        }
                                        if (mouse_x >= 500 && mouse_x <= 750 && mouse_y >= 425 && mouse_y <= 500)
                                        {
                                                m_window->Close();
                                                result = false;
                                        }
                                }
                        }
                        else if (!m_menu)
                        {
                                // Gestion du clic sur les feux
                                for (int i=0; i<m_model->getPartie()->getRoutes().size();i++)
                                {
                                        if (!m_model->getPartie()->getRoutes()[i]->getSens())
                                                for (int j=0; j < m_model->getPartie()->getRoutes()[i]->getListeDeCroisements().size() ; j++)
                                                        if (m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPresenceFeu())
                                                                if((event.Type == sf::Event::MouseButtonPressed) && (event.MouseButton.Button == sf::Mouse::Left
))
                                                                        if (mouse_x >= m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getPosi
tionX() - TAILLE_LARGEUR_FEU &&
                                                                                mouse_y >= m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->get
PositionY() - TAILLE_HAUTEUR_FEU &&
                                                                                mouse_x <= m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->get
PositionX() &&
                                                                                mouse_y <= m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->get
PositionY())
                                                                                        m_model->getPartie()->getRoutes()[i]->getListeDeCroisements()[j]->getFeu
()->changerCouleurFeu();
                                        }
                                }
                        }

                }
        return result;
}
```

```
#ifndef _GAMEVIEW_H_
#define _GAMEVIEW_H_

#include "GameModel.h"
#include <SFML/Audio.hpp>
#include <SFML/Graphics.hpp>
#include <sstream>
#include <fstream>
#include <string>


class GameView {
private:
        int m_width, m_height;
        bool m_menu, m_animation;

        sf::RenderWindow * m_window;
        sf::Font m_font;

        sf::Clock horloge;
        sf::View m_chargement;
        sf::View m_interNiveau;

        sf::Image m_image_logo_iut;
        sf::Image m_image_background;
        sf::Image m_image_vRoute;
        sf::Image m_image_hRoute;
        sf::Image m_image_croisement;
        sf::Image m_image_ligne;
        sf::Image m_image_feu_rouge;
        sf::Image m_image_feu_vert;
        sf::Image m_image_h_police;
        sf::Image m_image_h_voiture;
        sf::Image m_image_h_camion;
        sf::Image m_image_v_police;
        sf::Image m_image_v_voiture;
        sf::Image m_image_v_camion;
        sf::Image m_image_explosion;

        sf::Sprite m_sprite_logo_iut;
        sf::Sprite m_sprite_background;
        sf::Sprite m_sprite_vRoute;
        sf::Sprite m_sprite_hRoute;
        sf::Sprite m_sprite_croisement;
        sf::Sprite m_sprite_ligne;
        sf::Sprite m_sprite_feu;
        sf::Sprite m_sprite_police;
        sf::Sprite m_sprite_voiture;
        sf::Sprite m_sprite_camion;
        sf::Sprite m_sprite_h_police;
        sf::Sprite m_sprite_h_voiture;
        sf::Sprite m_sprite_h_camion;
        sf::Sprite m_sprite_v_police;
        sf::Sprite m_sprite_v_voiture;
        sf::Sprite m_sprite_v_camion;
        sf::Sprite m_sprite_explosion;

        GameModel * m_model;

        //Introduction
        sf::String m_intro;

        //Menu
        sf::String m_titre;
        sf::Image m_image_bouton_jouer;
        sf::Image m_image_bouton_quitter;
        sf::Sprite m_sprite_bouton_jouer;
        sf::Sprite m_sprite_bouton_quitter;

        //Jeu
        sf::Shape m_barre_traffic;
        sf::Shape m_barre_traffic_actuel;
        sf::String m_niveauEnCours;
        sf::String m_nbVehiculesObjectif;
        sf::String m_traffic;
        sf::String m_scoreActuel;
        sf::Shape m_barre_partie;
        std::string m_niveau;
        std::string m_score;
        std::string m_objectif;

        //Inter-Niveau
        sf::String m_texte_interNiveau_gagne;
        sf::String m_texte_interNiveau_perdu;
        std::string m_texte_gagne;
        std::string m_texte_perdu;


public:
        GameView(int width, int height);
        ~GameView();

        void setModel(GameModel * model);
        bool getMenu(GameView * view) const;

        std::string convertInt(int number);

        void ecranChargement();
        void ecranInterNiveau();
        void declarationImages();
        void draw();
        void affichageEcran();
        void affichageCroisement(int i);

        bool treatEvents();

        void afficherHighscore();
        void setHighscoreFichier();
};
#endif
```

```
#include <iostream>
#include <cstdlib>

#include "GameModel.h"
#include "GameView.h"
#include "Constantes.h"
#include "Vehicule.h"

using namespace std;
using namespace sf;

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


        GameModel * model = new GameModel(LARGEUR_FENETRE, HAUTEUR_FENETRE);
        GameView * view = new GameView(LARGEUR_FENETRE, HAUTEUR_FENETRE);
        view->setModel(model);

        while(view->treatEvents()){
                if (!view->getMenu(view))
                        model->nextStep();
                view->draw();
                usleep(11000);

        }

        delete view;
        delete model;


        return EXIT_SUCCESS;
}
```

```
CC=g++ -lsfml-graphics -lsfml-window -lsfml-system
OBJECTS=Voiture.o Camion.o Police.o Vehicule.o Feu.o Objet.o Route.o GameModel.o GameView.o Main.o Niveau.o Partie.o Croisement.o
TARGET=TrafficJam


${TARGET} : $(OBJECTS)
	$(CC) $(OBJECTS) -lsfml-graphics -lsfml-window -lsfml-system -o $(TARGET)

Voiture.o : Voiture.cc Voiture.h
	$(CC) -c Voiture.cc

Camion.o : Camion.cc Camion.h
	$(CC) -c Camion.cc

Police.o : Police.cc Police.h
	$(CC) -c Police.cc

Vehicule.o : Vehicule.cc Vehicule.h
	$(CC) -c Vehicule.cc

Feu.o : Feu.cc Feu.h
	$(CC) -c Feu.cc

Objet.o : Objet.cc Objet.h
	$(CC) -c Objet.cc

Route.o : Route.cc Route.h
	$(CC) -c Route.cc

GameModel.o : GameModel.cc GameModel.h
	$(CC) -c GameModel.cc

GameView.o : GameView.cc GameView.h
	$(CC) -c GameView.cc

Main.o : Main.cc GameModel.h GameView.h
	$(CC) -c Main.cc

Niveau.o : Niveau.cc Niveau.h
	$(CC) -c Niveau.cc

Partie.o : Partie.cc Partie.h
	$(CC) -c Partie.cc

Croisement.o : Croisement.cc Croisement.h
	$(CC) -c Croisement.cc

clean :
	rm -rf *.o $(TARGET)

# do not delete



#~ CXX=g++
#~ CXXFLAGS= -g -Wall -pedantic
#~ LDFLAGS= -lsfml-audio -lsfml-graphics -lsfml-window -lsfml-system
#~ EXEC = TrafficJam
#~ EXECPATH = .
#~ OBJ  = main.o GameModel.o GameView.o
#~
#~ all: $(EXEC)
#~
#~ TrafficJam: $(OBJ)
	#~ $(CXX) $(LDFLAGS) -o $(EXECPATH)/$@ $(OBJ)
#~
#~ main.o: main.cc GameView.h GameModel.h
	#~ $(CXX) -o $@ -c $< $(CXXFLAGS)
	#~
#~ GameView.o: GameView.cc GameView.h
	#~ $(CXX) -o $@ -c $< $(CXXFLAGS)
#~
#~ GameModel.o: GameModel.cc GameModel.h
	#~ $(CXX) -o $@ -c $< $(CXXFLAGS)
#~
#~ clean:
	#~ @rm -rf *.o $(EXECPATH)/$(EXEC)
```

```
#include "Niveau.h"
#include "Constantes.h"
#include "Croisement.h"


using namespace std;

//Constructeurs
Niveau::Niveau():m_nbRoutes(0), m_nbVehicules(0), m_nbFeux(0), m_trafficAutorise(0), m_collision(0), m_objectif(0), m_nbVehiculesPasses(0), m_nbVehiculePresent(
0)
{
        m_listeDeRoutes.push_back(new Route());
}
//(Editeur algorithmique de niveau)
Niveau::Niveau(int nbRoutes, int nbVehicules, int trafficAutorise, int nbFeux, int objectif): m_nbRoutes(nbRoutes), m_nbVehicules(nbVehicules), m_nbFeux(nbFeux)
,                                                                                             m_trafficAutorise(trafficAutorise), m_collision(0), m_objectif(objectif), m_nbVehicules
Passes(0), m_nbVehiculePresent(0)
{
        int nbRouteHorizontale=1, nbRouteVerticale=1, nbFeu=0;
        creerRoute();
        affecterSensEtCoordonneesRoute(nbRouteHorizontale, nbRouteVerticale);
        creerVehicule();
        affecterSensEtCoordonneesVehicules();
        creerCroisement(nbRouteHorizontale, nbRouteVerticale);
        affecterCoordonneeCroisement();
        creerFeu(nbRouteHorizontale, nbRouteVerticale, nbFeux);
}
//Destructeur
Niveau::~Niveau()
{
        for (int i=0; i<m_listeDeRoutes.size() ; i++)
        {
                delete m_listeDeRoutes[i];
                m_listeDeRoutes[i] = 0;
        }
}

//Accesseurs en lecture
int Niveau::getNbRoutes() const
{
        return m_nbRoutes;
}

int Niveau::getNbVehicules() const
{
        return m_nbVehicules;
}

int Niveau::getTrafficAutorise() const
{
        return m_trafficAutorise;
}

int Niveau::getNbFeux() const
{
        return m_nbFeux;
}

int Niveau::getObjectif() const
{
        return m_objectif;
}

bool Niveau::getCollision() const
{
        return m_collision;
}

int Niveau::getNbVehiculesPasses() const
{
        return m_nbVehiculesPasses;
}

int Niveau::getNbVehiculePresent() const
{
        return m_nbVehiculePresent;
}

vector <Route*> Niveau::getListeDeRoutes() const
{
        return m_listeDeRoutes;
}

//Accesseurs en Ã©criture
void Niveau::setNbRoutes(int nb)
{
        m_nbRoutes=nb;
}

void Niveau::setNbVehicules(int nb)
{
        m_nbVehicules=nb;
}

void Niveau::setTrafficAutorise(int nb)
{
        m_trafficAutorise=nb;
}
void Niveau::setNbFeux(int nb)
{
        m_nbFeux=nb;
}

void Niveau::setCollision(bool b)
{
        m_collision=b;
}

void Niveau::setNbVehiculesPasses(int nb)
{
        m_nbVehiculesPasses=nb;
}

void Niveau::setNbVehiculePresent(int nb)
{
        m_nbVehiculePresent = nb;
}

void Niveau::setObjectif(int nb)
{
        m_objectif=nb;
}

//Methodes
```



```
void Niveau:: creerRoute()
{
        for (int i=0; i<m_nbRoutes; i++)

                m_listeDeRoutes.push_back(new Route());
}

void Niveau:: creerVehicule()
{
        for (int i=0; i<m_nbRoutes; i++)

                for (int k=0; k<m_nbVehicules;k++)
                {
                        m_listeDeRoutes[i]->ajouterVehicule();
                                for (int l=0; l<m_listeDeRoutes[i]->getListeDeVehicules().size();l++)
                                        m_listeDeRoutes[i]->getListeDeVehicules()[l]->setSens(1);
                }
}

void Niveau:: creerCroisement(int nbRouteHorizontale, int nbRouteVerticale)
{
        for (int i=0; i<m_nbRoutes; i++)
        {
                if (m_listeDeRoutes[i]->getSens()==false)
                        for (int j=0; j<nbRouteVerticale-1; j++)
                        {
                                m_listeDeRoutes[i]->ajouterCroisement();
                                m_listeDeRoutes[i]->getListeDeCroisements()[j]->setPosY(m_listeDeRoutes[i]->getPositionY());
                        }
                else
                        for (int j=0; j<nbRouteHorizontale-1; j++)
                        {
                                m_listeDeRoutes[i]->ajouterCroisement();
                                m_listeDeRoutes[i]->getListeDeCroisements()[j]->setPosX(m_listeDeRoutes[i]->getPositionX());

                        }

        }
}

void Niveau:: creerFeu(int nbRouteHorizontale, int nbRouteVerticale, int nbFeux)
{
        int nb=0;
        for (int i=0; i<m_nbRoutes; i++)
        {
                if (m_listeDeRoutes[i]->getSens()==false)
                        for (int j=0; j<nbRouteVerticale-1; j++)
                        {
                                if (nb<nbFeux)
                                {
                                        m_listeDeRoutes[i]->getListeDeCroisements()[j]->setPresenceFeu(1);
                                        m_listeDeRoutes[i]->getListeDeCroisements()[j]->genererFeu();
                                        nb++;
                                }
                        }
        }
}

void Niveau:: affecterSensEtCoordonneesRoute(int &nbRouteHorizontale, int &nbRouteVerticale)
{
        for (int i=0; i<m_nbRoutes; i++)

        {
                if (i%2==1)
                        m_listeDeRoutes[i]->setSens(1);
                switch (m_listeDeRoutes[i]->getSens())
                {
                        case 0:
                        {
                                m_listeDeRoutes[i]->setPosY(nbRouteVerticale*ECART_VERTICAL_ROUTES);
                                nbRouteHorizontale++;
                                break;
                        }
                        case 1:
                        {
                                m_listeDeRoutes[i]->setPosX((nbRouteHorizontale-1)*ECART_HORIZONTAL_ROUTES);
                                nbRouteVerticale++;
                                break;
                        }
                        default : break;
                }
        }
}

void Niveau:: affecterSensEtCoordonneesVehicules()
{
        for (int i=0; i<m_nbRoutes; i++)

                for (int j=0; j<m_listeDeRoutes[i]->getListeDeVehicules().size();j++)
                        switch (m_listeDeRoutes[i]->getSens())
                        {
                                case 0:
                                {
                                        m_listeDeRoutes[i]->getListeDeVehicules()[j]->setSens(0);
                                        m_listeDeRoutes[i]->getListeDeVehicules()[j]->setPosX(m_listeDeRoutes[i]->getPositionX()-128);
                                        m_listeDeRoutes[i]->getListeDeVehicules()[j]->setPosY(m_listeDeRoutes[i]->getPositionY()+5);
                                        break;
                                }
                                case 1:
                                {
                                        m_listeDeRoutes[i]->getListeDeVehicules()[j]->setSens(1);
                                        m_listeDeRoutes[i]->getListeDeVehicules()[j]->setPosX(m_listeDeRoutes[i]->getPositionX()+5);
                                        m_listeDeRoutes[i]->getListeDeVehicules()[j]->setPosY(m_listeDeRoutes[i]->getPositionY()-128);
                                        break;
                                }
                                default: break;
                        }
}

void Niveau:: affecterCoordonneeCroisement()
{
        for (int i=0; i<m_nbRoutes; i++)
        {
                if (m_listeDeRoutes[i]->getSens() == false)
                        for (int j=0 ; j <= m_listeDeRoutes[i]->getListeDeCroisements().size() ; j++)
                                if (m_listeDeRoutes[j]->getSens()==true)
                                        for (int k=0; k<m_listeDeRoutes[i]->getListeDeCroisements().size();k++)
                                                m_listeDeRoutes[i]->getListeDeCroisements()[k]->setPosX((k+1)*ECART_HORIZONTAL_ROUTES);
                if (m_listeDeRoutes[i]->getSens() == true)
                        for (int j=0 ; j <= m_listeDeRoutes[i]->getListeDeCroisements().size() ; j++)
                                if (m_listeDeRoutes[j]->getSens()==false)
                                        for (int k=0; k<m_listeDeRoutes[i]->getListeDeCroisements().size();k++)
                                                m_listeDeRoutes[i]->getListeDeCroisements()[k]->setPosY((k+1)*ECART_VERTICAL_ROUTES);
        }
}
```

```
#ifndef _NIVEAU_H_
#define _NIVEAU_H_

#include <vector>
#include "Route.h"

class Niveau
{
        private:
                int m_nbRoutes;
                int m_nbVehicules;
                int m_nbFeux;
                int m_trafficAutorise;
                bool m_collision;
                int m_objectif;
                int m_nbVehiculesPasses;
                int m_nbVehiculePresent;
                std::vector <Route*> m_listeDeRoutes;
        public:
                Niveau();
                Niveau(int nbRoutes, int nbVehicules, int trafficAutorise, int nbFeux, int objectif);
                ~Niveau();
                std::vector <Route*> getListeDeRoutes() const;
                int getNbRoutes() const;
                int getNbVehicules() const;
                int getTrafficAutorise() const;
                int getNbFeux() const;
                int getObjectif() const;
                bool getCollision() const;
                int getNbVehiculesPasses() const;
                int getNbVehiculePresent() const;
                void setNbRoutes(int nb);
                void setNbVehicules(int nb);
                void setTrafficAutorise(int nb);
                void setNbFeux(int nb);
                void setCollision(bool b);
                void setNbVehiculesPasses(int nb);
                void setObjectif(int nb);
                void setNbVehiculePresent(int nb);
                void creerRoute();
                void creerVehicule();
                void creerCroisement(int nbRouteHorizontale, int nbRouteVerticale);
                void creerFeu(int nbRouteHorizontale, int nbRouteVerticale, int nbFeu);
                void affecterSensEtCoordonneesRoute(int &nbRouteHorizontale, int &nbRouteVerticale);
                void affecterSensEtCoordonneesVehicules();
                void affecterCoordonneeCroisement();
};

#endif
```

```
#include "Objet.h"
#include <cstdlib>

using namespace std;

//Constructeurs
Objet::Objet():m_posX(0), m_posY(0), m_estVertical(0)
{}
Objet::Objet(float posX, float posY):m_posX(posX), m_posY(posY), m_estVertical(0)
{}
Objet::Objet(float posX, float posY, bool estVertical):m_posX(posX), m_posY(posY), m_estVertical(estVertical)
{}

//Destructeur
Objet::~Objet()
{}

//Accesseurs en lecture
float Objet::getPositionX() const
{
        return m_posX;
}

float Objet::getPositionY() const
{
        return m_posY;
}

bool Objet::getSens() const
{
        return m_estVertical;
}

//Accesseurs en Ã©criture
void Objet::setPosX(float posX)
{
        m_posX=posX;
}

void Objet::setPosY(float posY)
{
        m_posY=posY;
}

void Objet::setSens(bool estVertical)
{
        m_estVertical=estVertical;
}
```

```
#ifndef _OBJET_H_
#define _OBJET_H_

class Objet
{
        protected :
                float m_posX;
                float m_posY;
                bool m_estVertical;
        public:
                Objet();
                Objet(float posX, float posY);
                Objet(float posX, float posY, bool estVertical);
                ~Objet();
                float getPositionX() const;
                float getPositionY() const;
                bool getSens() const;
                void setPosX(float posX);
                void setPosY(float posY);
                void setSens(bool estVertical);
};

#endif
```

```
#include "Partie.h"
#include "Constantes.h"

using namespace std;

//Constructeur
Partie::Partie(): m_score(0), m_niveauEnCours(1)
{
	m_niveau=new Niveau(1, 15, 13, 1, 5);
}

//Destructeur
Partie::~Partie()
{
	delete m_niveau;
}

//Accesseurs en lecture
int Partie::getScore() const
{
	return m_score;
}

Niveau * Partie::getNiveau() const
{
	return m_niveau;
}

int Partie::getNiveauEnCours() const
{
	return m_niveauEnCours;
}

vector <Route*> Partie::getRoutes()
{
	return m_niveau->getListeDeRoutes();
}
//Accesseurs en Ã©criture
void Partie::setScore(int nb)
{
	m_score+=nb;
}

void Partie::setNiveauEnCours(int nb)
{
	m_niveauEnCours=nb;
}

//MÃ©thodes
void Partie::genererNiveau()
{
	m_niveau=new Niveau(calculerNbRoutes(), calculerNbVehicules(), calculerTrafficAutorise(),calculerNbFeux(), calculerObjectif());
}

void Partie:: supprimerNiveau()
{
	delete m_niveau;
}

int Partie::calculerNbRoutes()
{
	if (m_niveauEnCours<4)
		return m_niveauEnCours+1;
	else if (m_niveauEnCours>=4 && m_niveauEnCours<NB_ROUTES_MAX)
			return m_niveauEnCours;
	else
		return NB_ROUTES_MAX;
}

int Partie::calculerNbVehicules()
{
	return m_niveauEnCours*15;
}

int Partie::calculerTrafficAutorise()
{
	return (m_niveau->getObjectif() + m_niveau->getNbVehicules());
}

int Partie:: calculerNbFeux()
{
	return m_niveauEnCours;
}

int Partie:: calculerScore(int i, int j)
{
	switch (getRoutes()[i]->getListeDeVehicules()[j]->getVitesse())
	{
		case 1 : return 3; break;
		case 2 : return 1; break;
		case 3 : return 1; break;
	}
}

int Partie:: calculerObjectif()
{
	return m_niveauEnCours*5;
}
```

```
#ifndef _PARTIE_H_
#define _PARTIE_H_

#include <vector>
#include "Niveau.h"

class Partie
{
        private:
                int m_score;
                int m_niveauEnCours;
                Niveau * m_niveau;
        public:
                Partie();
                ~Partie();
                int getScore() const;
                int getNiveauEnCours() const;
                void setScore(int nb);
                void setNiveauEnCours(int nb);
                Niveau * getNiveau() const;
                void genererNiveau();
                void supprimerNiveau();
                int calculerNbRoutes();
                int calculerNbVehicules();
                int calculerTrafficAutorise();
                int calculerNbFeux();
                int calculerScore(int i, int j);
                int calculerObjectif();
                std ::vector <Route*> getRoutes();
};

#endif
```

```
#include "Police.h"

using namespace std;

//Contructeur
Police::Police(): Vehicule(), m_vitesse(3)
{}

//Destructeur
Police::~Police()
{}

//Accesseur en lecture
int Police::getVitesse() const
{
        return m_vitesse;
}
```

```
#ifndef _POLICE_H_
#define _POLICE_H_

#include "Vehicule.h"

class Police: public Vehicule
{
        private :
                int m_vitesse;
        public :
                Police();
                ~Police();
                int getVitesse() const;
};

#endif
```

```
#include "Route.h"
#include "Voiture.h"
#include "Camion.h"
#include "Police.h"
#include <SFML/System.hpp>

using namespace std;
using namespace sf;

//Constructeurs
Route::Route():Objet(), m_listeDeVehicules(0), m_listeDeCroisements(0), m_numeroVehicule(0)
{}

//Destructeur
Route::~Route()
{
        for (int i=0; i<m_listeDeVehicules.size() ; i++)
        {
                delete m_listeDeVehicules[i];
                m_listeDeVehicules[i] = 0;
        }
        for (int i=0; i<m_listeDeCroisements.size(); i++)
        {
                delete m_listeDeCroisements[i];
                m_listeDeCroisements[i]=0;
        }
}

//Accesseurs en lecture
vector <Vehicule*> Route::getListeDeVehicules() const
{
        return m_listeDeVehicules;
}

vector <Croisement*> Route::getListeDeCroisements() const
{
        return m_listeDeCroisements;
}

int Route::getNumeroVehicule() const
{
        return m_numeroVehicule;
}

//Accesseur en Ã©criture
void Route::setNumeroVehicule (int nb)
{
        m_numeroVehicule = nb;
}

//Methodes
void Route::ajouterVehicule()
{
        int nbAlea=Randomizer::Random(1,8);
        if (1<=nbAlea && nbAlea<=6)
                m_listeDeVehicules.push_back(new Voiture());
        else if (nbAlea==7)
                m_listeDeVehicules.push_back(new Camion());
        else
                m_listeDeVehicules.push_back(new Police());
}

void Route::ajouterCroisement()
{
                m_listeDeCroisements.push_back(new Croisement());
}
```

```
#ifndef _ROUTE_H_
#define _ROUTE_H_

#include <vector>
#include "Vehicule.h"
#include "Croisement.h"

class Route : public Objet
{
        private :
                std::vector <Vehicule*> m_listeDeVehicules;
                std::vector <Croisement*> m_listeDeCroisements;
                int m_numeroVehicule;
        public :
                Route();
                ~Route();
                std::vector <Vehicule*> getListeDeVehicules() const;
                std::vector <Croisement*> getListeDeCroisements() const;
                int getNumeroVehicule() const;
                void setNumeroVehicule (int nb);
                void ajouterVehicule();
                void ajouterCroisement();
};

#endif
```

```
#include "Vehicule.h"

using namespace std;


//Constructeur
Vehicule::Vehicule(): Objet(), m_roule(0)
{}

//Destructeur
Vehicule::~Vehicule()
{}

//Accesseurs en lecture

bool Vehicule::getEtatVehicule() const
{
        return m_roule;
}

void Vehicule::setEtatVehicule(bool etat)
{
        m_roule = etat;
}
//Methodes
void Vehicule::deplacement()
{
        if(m_roule)
                switch (m_estVertical)
                {
                        case 0: m_posX+=getVitesse(); break;
                        case 1: m_posY+=getVitesse(); break;
                }
}

int Vehicule::getVitesse() const
{
        return getVitesse();
}
```

```
#ifndef _VEHICULE_H_
#define _VEHICULE_H_

#include "Objet.h"

class Vehicule: public Objet
{
        protected :
                bool m_roule;
        public :
                Vehicule();
                virtual ~Vehicule();
                bool getEtatVehicule() const;
                void setEtatVehicule(bool etat);
                virtual int getVitesse() const;
                void deplacement();

};

#endif
```

```
#include "Voiture.h"

using namespace std;

//Contructeur
Voiture::Voiture(): Vehicule(), m_vitesse(2)
{}

//Destructeur
Voiture::~Voiture()
{}

//Accesseur en lecture
int Voiture::getVitesse() const
{
        return m_vitesse;
}
```

```
#ifndef _VOITURE_H_
#define _VOITURE_H_

#include "Vehicule.h"

class Voiture: public Vehicule
{
        private :
                int m_vitesse;
        public :
                Voiture();
                ~Voiture();
                int getVitesse() const;
};

#endif
```