Rapport du Projet

Bibliothèques de Développement Multimédia

WU Jinda
ZHOU Nan

1. Les spécifications de l'interface utilisateur

Le but du projet est de développer une application pour un jeu de conduite automobile. Les utilisateurs doivent contrôler la voiture à l'aide des touches fléchées sur les claviers ou des gestes à l'aide d'une webcam pour éviter les véhicules venant en sens inverse. Lorsque vous utilisez des claviers, vous pouvez contrôler le mouvement de la voiture en appuyant sur les touches fléchées gauche et droite. Et vous pouvez mettre en pause et continuer le jeu en appuyant sur la barre d'espace.

Gestes contrôlés par caméra:

1. aller tout droit à vitesse normale : les mains sont en position neutre ;
2. tourner à droite : dans un geste de rotation vers la droite, la main gauche monte alors que la main droite descend, le geste est actif tant que la main gauche est plus haute que la main droite;
3. tourner à gauche : dans un geste de rotation vers la gauche, la main droite monte alors que la main gauche descend, le geste est actif tant que la main droite est plus haute que la main gauche ;
4. arrêt : depuis la position neutre, les deux poings se rapprochent jusqu'à quelques centimètres, les mains étant toutes les deux au même niveau, l'action est réalisée dès que les poings sont suffisamment proches.

Lorsque les deux mains quittent la zone de détection de la caméra, l'état de pause est également activé.

(*Lorsque la caméra détecte un seul poing, le rectangle s'affiche en rouge et le programme le considère comme une erreur de détection et conserve l'état actuel.*)

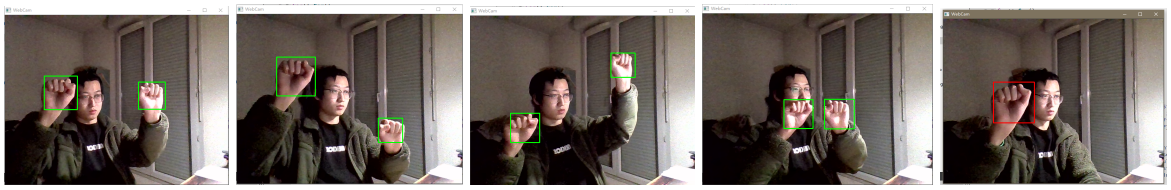

Figure 1--Fenêtre de webcam

Afin de jouer plus longtemps et d'obtenir un score plus élevé, les utilisateurs doivent ramasser des bidons d'essence pour faire le plein de la voiture en cliquant sur les bidons d'essence à l'écran. Lorsqu'une collision se produit, le jeu est terminé et une boîte de dialogue apparaît qui affiche le temps de jeu de l'utilisateur et le score obtenu par l'utilisateur. L'utilisateur peut choisir de redémarrer un nouveau jeu ou de quitter le jeu.

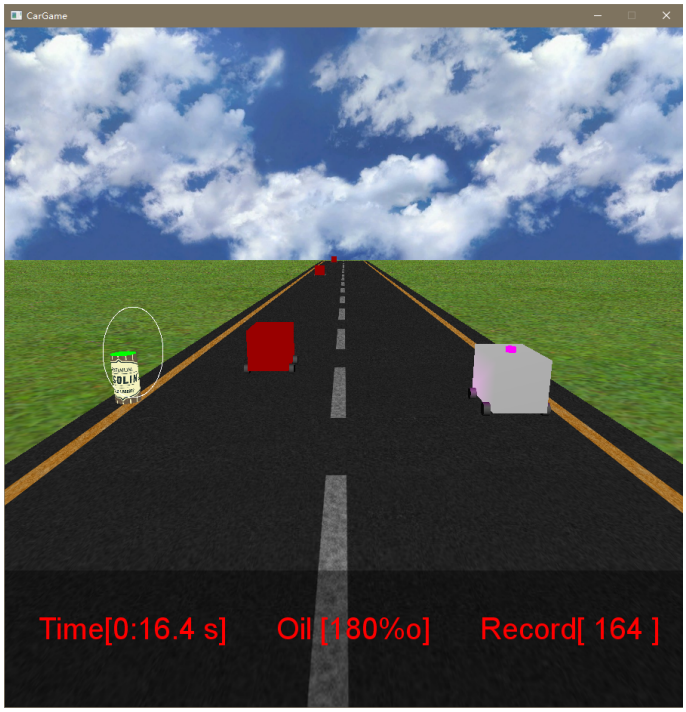

Figure 2--Fenêtre de Cargame

Figure 3--Boîte de Dialogue

L'application est développée dans QT Creator. Nous créons l'interface de jeu avec la bibliothèque OpenGL. Et nous traitons la vidéo obtenue par webcam avec la librairie OpenCV.

Le lien du code source et des fichiers exécutables sur Google drive:
https://drive.google.com/drive/folders/1oqOVW4aXB0ZrXDN2zQuO_mV27xF6vW0U?usp=sharing

2. Les principales classes

- **Main function**

  Générez et gérez l'ensemble de l'application. Il inclut la classe «MyGLWidget» et la classe «VideoProcessing» et les réalise.

- **Class 'MyGLWidget'**

  Fenêtre principale de l'application. Il inclut toutes les autres classes à l'exception de la classe «VideoProcessing» et réalise ces classes. Il hérite de la classe QOpenGLWidget. Il dessine l'interface du jeu, active les systèmes de lumière et de profondeur, définit le mode d'affichage et la position de la caméra, définit les événements du clavier et de la souris, etc.

- **Class 'Car'**

  Il enregistre les coordonnées, les matériaux et les informations sur le carburant de la voiture.

- **Class ‘Ground’**

  Il enregistre les coordonnées, les matériaux et les informations de texture de la route, du ciel, de l'herbe et du bidon.

- **Class ‘OppCars’**

  Il enregistre les coordonnées et les informations sur les matériaux d'une voiture opposée.

- **Class ‘OppCarVector’**

  Il construit un vecteur qui doit stocker chaque voiture opposée à l'écran.

- **Class ‘Dialog’**

  Il gère l'interface utilisateur de la boîte de dialogue. Il contient les commandes des boutons «restart» et «quit».

- **Class ‘VideoProcessing’**

  Il contient des méthodes de traitement du streaming vidéo, de détection des poings et des paumes et de détection des mouvements des mains à l'aide de la bibliothèque OpenCV.

Voici la diagramme de class:

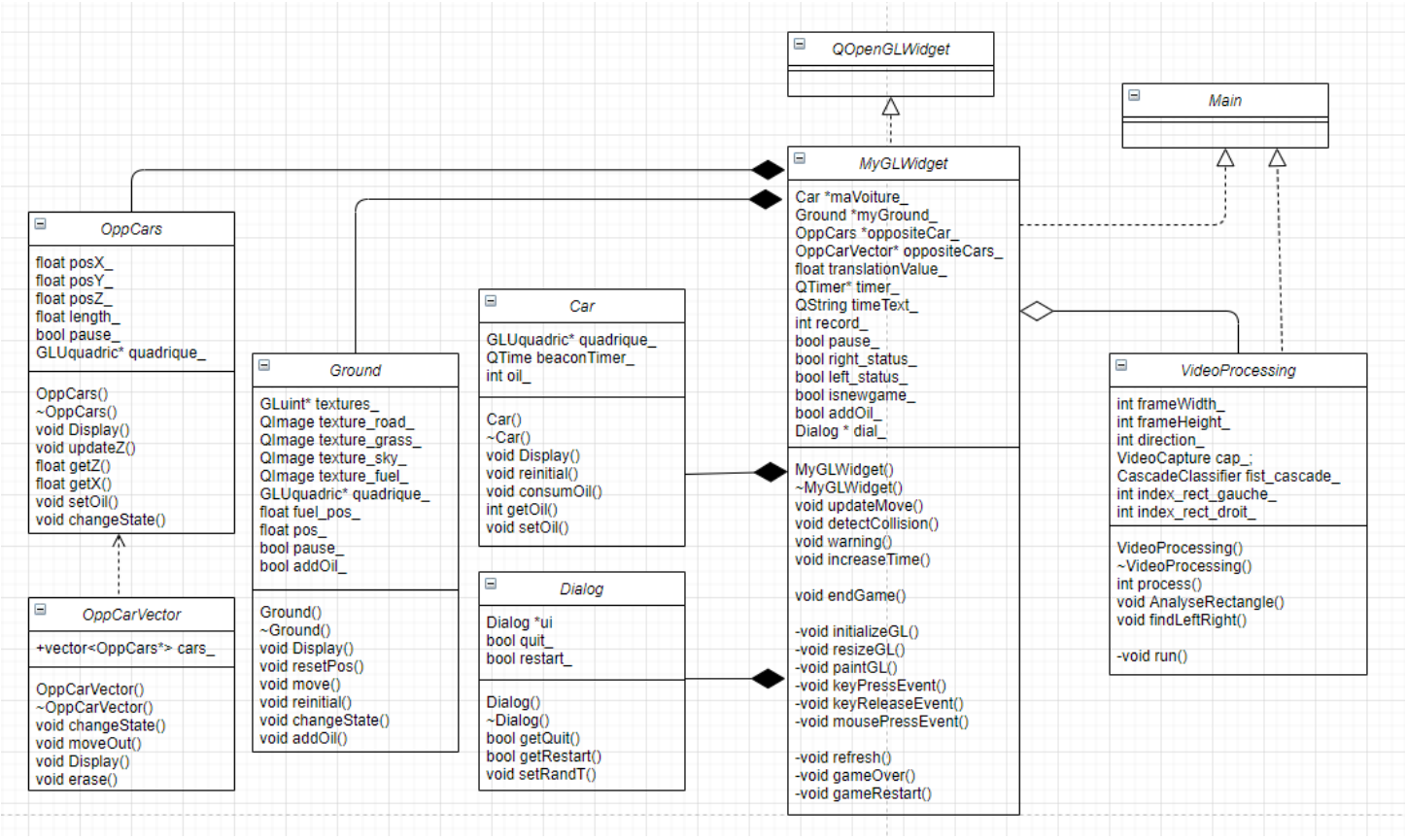

Figure 4--Diagramme des classes

3. L'état de finalisation de l'application

Fonctionnalités à implémenter:

1. affichage de la scène 3D avec la voiture, la route, l'herbe, le ciel ainsi que les éclairages et les textures ; **Réalisé**
2. déplacements latéraux de la voiture commandés par les gestes de rotation des mains, présence d'une zone affichant l'image de la webcam en continu ; **Réalisé**
3. apparition de bidons d'essence texturés et possibilité de cliquer avec la souris sur un bidon pour récupérer l'essence correspondante ; **Réalisé**
4. détection du geste de rapprochement des mains, gestion de l'arrêt de la voiture et de son re-démarrage lorsque les mains sont revenues en position neutre, matérialisation d'une zone d'arrêt sur la route ; **Réalisé**
5. apparition de véhicules en sens inverse et gestion de leur déplacement ; **Réalisé**
6. gestion des collisions avec les autres véhicules et avec le bord de la route ; **Réalisé**
7. affichage du chronomètre et de la jauge d'essence, gestion de leur défilement ; **Réalisé**
8. détection de la fin de la partie, affichage du score et ré-initialisation pour commencer une nouvelle partie. **Réalisé**

Fonctions non réalisées et problèmes existants:

- L'intégration des jeux de voiture et de la webcam dans l'interface QT n'a pas été implémentée
- Lorsque la webcam fonctionne normalement, vous ne pouvez pas terminer le programme en cliquant sur le bouton Quitter dans la boîte de dialogue.

Au lieu de cela, vous devez appuyer sur la touche «Échap» du clavier pour contrôler l'extrémité de la webcam, puis cliquer sur la croix en dix dans le coin supérieur droit pour quitter le programme.

4. Les fichiers d'entête des classes

Car.h

```
#ifndef CAR_H
#define CAR_H
#include <qopengl.h>
#include <GL/glu.h>
#include <QTime>
/*
* Cette classe vise à dessiner les véhicules
* et les éléments associés contrôlés par le joueur.
*/
class Car  // auteurs : WU & ZHOU
{

public:

  Car();// Constructeur
  ~Car();// Destructeur
  void Display();// Methode d'affichage
  void reinitial();// Methode d'initialisation
  void consumeOil();// Methode de Consommation d'essence
  int getOil() {return oil_;};//Obtenir la quantité de carburant actuelle
```

```
    void setOil(int oil){oil_ = oil;};//Régler la quantité de carburant

private:
    GLUquadric* quadrique_ = nullptr;
    QTime beaconTimer_;
    int oil_;
};
#endif // CAR_H
```

dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>

/*
* Cette classe vise à générer des dialogues pour interagir avec les utilisateurs.
*/
namespace Ui {
class Dialog;
}

class Dialog : public QDialog  // auteurs : WU & ZHOU
{
    Q_OBJECT

public:
```

```
    explicit Dialog(QWidget *parent = nullptr);
    ~Dialog();

    bool getQuit() {return quit_;}//Vérifiez si l'utilisateur souhaite se déconnecter
    bool getRestart() {return restart_;}//Vérifiez si l'utilisateur souhaite
recommencer
    void setRandT(int record, QString time);//Le temps et le score sont affichés
dans la boîte de dialogue.
private:
    Ui::Dialog *ui;
    bool quit_ = false;
    bool restart_ = false;

private slots:
    void on_Restart_clicked();
    void on_Quit_clicked();
};

#endif // DIALOG_H
```

Ground.h

```
#ifndef GROUND_H
#define GROUND_H
#include <qopengl.h>
#include <QImage>
#include <GL/glu.h>
#include <QDebug>
```

```
/*
* Cette classe vise à dessiner des éléments liés à la terre
* et à gérer la lumière ambiante et les textures d'objets.
*/
class Ground   // auteurs : WU & ZHOU
{
//    Q_OBJECT;
public:

   Ground();// Constructeur
   ~Ground();// Destructeur
   void Display();// Methode d'affichage

   void resetPos();//Réinitialiser la position
   void move();//changer la position
   void reinitial();//Réinitialiser tout les values de Ground
   void changeState(bool pause);//Modifiez s'il faut arrêter l'état.
   void addOil();//Vérifiez s'il faut faire le plein

private:
   GLuint* textures_=nullptr;
   QImage texture_road_;
   QImage texture_grass_;
   QImage texture_sky_;
   QImage texture_fuel_;
   GLUquadric* quadrique_ = nullptr;

   float fuel_pos_ = -198.5f;
   float pos_ = 0.0f;
```

```
    bool pause_ = false;
    bool addOil_ = false;

};
#endif // GROUND_H
```

myglwidget.h

```
#include "Car.h"
#include "Ground.h"
#include "oppcars.h"
#include "dialog.h"
#include <QOpenGLWidget>
#include <QKeyEvent>
#include <QTimer>
#include <QPainter>

/*
* Cette classe vise à afficher des éléments tels que les véhicules,
* le ciel et la terre dans la scène opengl, et exécute la logique de
* contrôle de la scène rafraîchissante en surveillant les événements
* de la souris et du clavier.
*/
class MyGLWidget : public QOpenGLWidget // auteurs : WU & ZHOU
{
    Q_OBJECT
public:
```

```
    // Constructeur
    MyGLWidget(QWidget * parent = nullptr);
    ~MyGLWidget();

    void updateMove(int move);// Fonction de changement de mouvement en
détectant les poings
    void detectCollision();//Fonction de détection de collision
    void warning();//Fonction pour faire apparaître une fenêtre de dialogue.
    void increaseTime();//Fonction d'augmentation du temps.
signals:
    void endGame();

protected:

    // Fonction d'initialisation
    void initializeGL();

    // Fonction de redimensionnement
    void resizeGL(int width, int height);

    // Fonction d'affichage
    void paintGL();

    // Fonction de gestion d'interactions clavier
    void keyPressEvent(QKeyEvent* event);
    void keyReleaseEvent(QKeyEvent * event);
    //Fonction de gestion d'interactions Souris
    void mousePressEvent(QMouseEvent *event);
```

```
    Dialog * dial_;

protected slots:
    void refresh();
    void gameOver();
    void gameRestart();

private:
    Car *maVoiture_ = nullptr;
    Ground *myGround_ = nullptr;
    OppCars *oppositeCar_ = nullptr;
    OppCarVector* oppositeCars_ = nullptr;

    float translationValue_ = 0.0f;

    QTimer* timer_ = nullptr;
    QString timeText_ = "00:00";
    int record_ = 0;

    bool pause_ = false;
    bool right_status_ = false;
    bool left_status_ = false;
    bool isnewgame_ = false;
    bool addOil_ = false;

};
```

oppcars.h

```
#ifndef OPPCARS_H
#define OPPCARS_H

#include <qopengl.h>
#include <GL/glu.h>
#include <time.h>
#include <vector>

using namespace std;
/*
* Cette classe est pour l'individu de la voiture venant en sens inverse.
*/
class OppCars // auteurs : WU & ZHOU
{
public:
  OppCars();// Constructeur
  ~OppCars();// Destructeur

  void Display();// Methode d'affichage
  void updateZ();// Mettre à jour l'emplacement
  float getZ();// Obtenez la coordonnée z
  float getX();// Obtenez la coordonnée x
  void changeState(bool b);// Modifiez s'il faut arrêter l'état.

private:
  float posX_ = 0.f;
  float posY_ = 0.f;
  float posZ_ = -198.5f;
```

```
    float length_ = 250.f;
    bool pause_ = false;

    GLUquadric* quadrique_ = nullptr;
};
/*
* Cette classe est destinée à la file d'attente des véhicules venant du côté
opposé.
*/
class OppCarVector  // auteurs : WU & ZHOU
{
public:
    vector<OppCars*> cars_;

    OppCarVector();// Constructeur
    OppCarVector(OppCars* car);// Constructeur avec parametres
    ~OppCarVector();// Destructeur

    void changeState(bool b); // Modifiez s'il faut arrêter l'état.
    void moveOut();// La méthode de contrôle de la voiture opposée pour sortir de
la frontière.
    void Display();// Methode d'affichage
    void erase(unsigned int);//Relâchez la voiture du côté opposé qui sort de la
frontière.
};

#endif // OPPCARS_H
```

videoprocessing.h

```
#ifndef VIDEOPROCESSING_H
#define VIDEOPROCESSING_H

#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <opencv2/objdetect.hpp>
#include <cstdio>
#include <iostream>
#include <cmath>
#include <QDebug>

using namespace cv;
using namespace std;
/*
* Ce class est destiné au traitement des flux vidéo,
* à l'analyse des images et à l'identification des poings.
*/
class VideoProcessing  // auteurs : WU & ZHOU
{
public:
  VideoProcessing();// Constructeur
  VideoProcessing(int frameWidth, int frameHeight);// Constructeur avec
parametres
  ~VideoProcessing();// Destructeur
  int process();//La méthode de traitement du flux vidéo.
  void AnalyseRectangle(std::vector<cv::Rect> listRect,bool choix);//La méthode
d'analyse des gestes.
```

```
    void findLeftRight(std::vector<cv::Rect> listRect);//La méthode de détermination
la main droite et de la main gauche.

protected:
        void run();

private:
    int frameWidth_=640;
    int frameHeight_=480;
    int direction_=0 ; // 0 : arrêt, 1: gauche, 2: droite

    VideoCapture cap_;
    CascadeClassifier fist_cascade_;
    int index_rect_gauche_,index_rect_droit_;
};

#endif // VIDEOPROCESSING_H
```